Titre: Title:	Une méthode d'estimation de la consommation de puissance pour un système sur puce
Auteur: Author:	Michel Rogers-Vallée
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Rogers-Vallée, M. (2012). Une méthode d'estimation de la consommation de puissance pour un système sur puce [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/846/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/846/
Directeurs de recherche: Advisors:	Guy Bois, & Marc-André Cantin
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

UNE MÉTHODE D'ESTIMATION DE LA CONSOMMATION DE PUISSANCE POUR UN SYSTÈME SUR PUCE

MICHEL ROGERS-VALLÉE
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

UNE MÉTHODE D'ESTIMATION DE LA CONSOMMATION DE PUISSANCE POUR UN SYSTÈME SUR PUCE

présenté par : ROGERS-VALLÉE, Michel
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury d'examen constitué de :

Mme. NICOLESCU, Gabriela, Doct., présidente.
M. BOIS, Guy, Ph.D., membre et directeur de recherche.
M. CANTIN, Marc-André, Ph.D., membre et codirecteur de recherche.
M. BELTRAME, Giovanni, Ph.D., membre.

If you don't know what to do... go up !
- adage d'escalade

REMERCIEMENTS

Je tiens tout d'abord à remercier Guy Bois de m'avoir offert la chance de faire cette maîtrise. Marc-André Cantin de m'avoir conseillé et supporté au cours de ma maîtrise. Ma famille, mes amis et mon amoureuse m'ont été d'un grand soutien moral pour passer au travers de ce travail. Et un coup de chapeau à mes partenaires d'escalade pour m'avoir donnés un moyen de changer mes idées quand j'en avais de besoins.

RÉSUMÉ

Estimer la consommation de puissance le plus tôt possible durant le cycle de développement est important pour pouvoir rencontrer le temps de mise en marché. Pour cela, plusieurs recherches en consommation de puissance se tournent vers l'estimation à haut niveau, comme la Modélisation au Niveau Transactionnel (TLM), pour accélérer l'obtention des estimations de puissance. Ce travail présente une méthodologie à haut-niveau orienté sur les Coeur sous licence (IP) qui effectue une estimation de puissance. La méthode propose une distinction entre l'activité de l'IP concerné et de son implémentation. Ceci permet de facilement créer un modèle qui peut être réutilisé avec différentes fréquences et implémentations. En utilisant l'information obtenue par des mesures d'une description au Niveau Registre (RTL), un modèle peut-être créé pour des simulations haut-niveau permettant d'abstraire l'implémentation. La méthodologie est présentée sur un processeur, une mémoire, un bus, une minuterie et un Contrôleur d'Interruption de Processeur (PIC) de Xilinx. En comparaison à des estimations effectuées au niveau RTL, le modèle permet d'estimer la consommation de puissance avec une précision de $25\% \pm 10\%$ par rapport à une estimation effectuée avec Xpower ; et ce avec un facteur accélération de trois ou quatre ordres de grandeur.

ABSTRACT

Estimating the power consumption of System on Chip as early as possible in the design life cycle is important to meet the time to market requirements. For this purpose, most research is turning toward high-level models, like the Transaction-Level Modeling (TLM), to estimate power earlier. This work presents a high-level Intellectual Property core (IP) oriented power estimation methodology. The methodology separates the activity of the IP from the implementation. This allows the ability to easily create a model that can be used with different frequencies, layout and implementation technology. By using data gathered from the Register-Transfer Level (RTL) a model can be created for high-level simulation that can take into account the technology and characteristics of the Field-Programmable Gate Array (FPGA) device. The methodology is presented in this work for a processor, its local memory IP, counter, Processor Interrupt Controller (PIC) and bus from Xilinx. Compared to estimations made at the RTL level, the resulting model gives accurate results of $25\% \pm 10\%$ compared to a Xpower estimate with three to four order speedups and through different implementations.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ALU	Unité Arithmétique et Logique
ASIC	Circuit Intégré à Aplication Spécifique
ASP	Paramètres Structurel Actif
BRAM	«Block Random Access Memory»
DUT	Circuit sous Test
FFT	Transformée de Fourier Rapide
FPGA	«Field Programmable Gate Array»
FPU	«Floating Point Unit»
FSM	Machine à États Finis
HDL	Langage de Description Matériel
I/O	Entrée et Sortie
IDE	Environnement de Développement Intégré
IP	Coeur sous licence
LMB	«Local Memory Bus»
MHS	Spécification Matériel du Processeur
MPSoC	«Multiprocessor System-on-Chip»
OPB	«On-Chip Peripheral Bus»
PAR	Placement et Routage
PC	Compteur de Programme
PIC	Contrôleur d'Interruption de Processeur
PMU	Unité de gestion de puissance
RAM	Mémoire Vive
ROM	Mémoire Morte
RTL	«Register Transfert Level»
SOC	Système sur Puce
SSP	Paramètres Structurel Statique
TLM	Modélisation au Niveau Transactionnel
TP	Paramètres Technologique
UART	«Universal asynchronous receiver/transmitter»
VCC	Voltage Positif
VCD	Registre de Transition de Variable

CHAPITRE 1

INTRODUCTION

Les projets en microélectronique accordent de plus en plus d'importance à la consommation de puissance des systèmes. La raison est que le concepteur matériel fait face à des sources de puissance limitée, que ce soit une prise électrique ou une batterie dans le cas de système embarqué. Puisque les systèmes augmentent en surface de silicium et en nombre de composantes, leur consommation augmente proportionnellement (la loi de Moore, voir Figure 1.1a). Par contre, la puissance disponible n'augmente pas au même rythme (voir la Figure 1.1b).

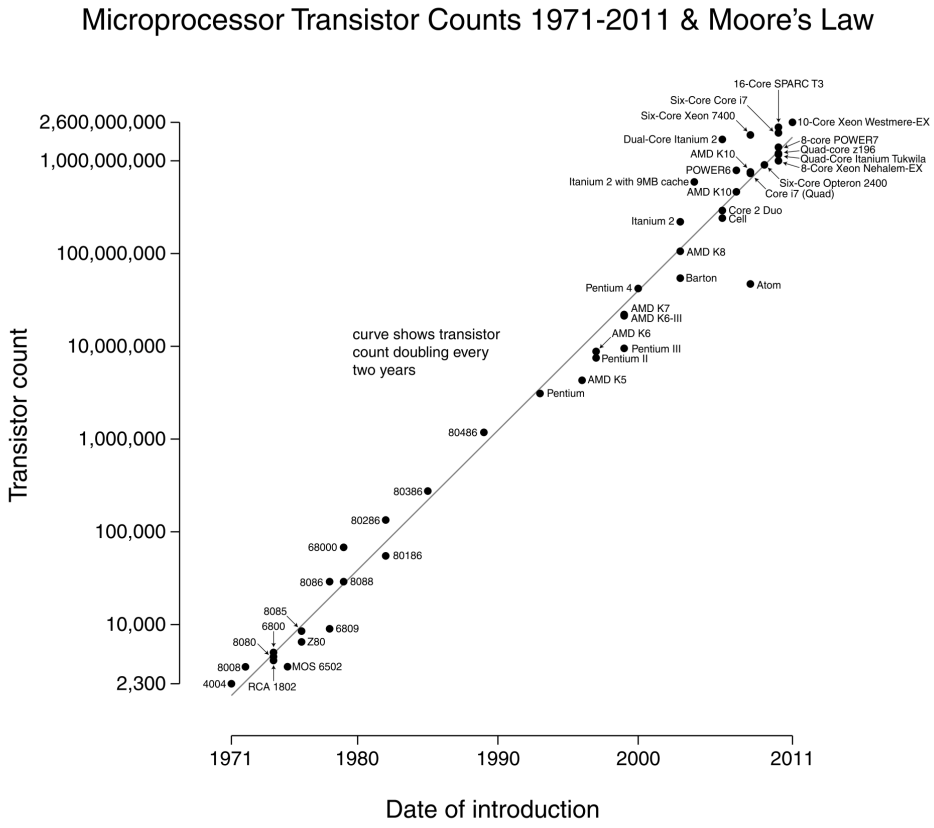
En ce moment, le nombre de transistor est doublé à tous les deux ans pour une même unité de surface de silicium. Par contre, la densité d'énergie des batteries augmente beaucoup plus lentement en doublant au 15 ans. Ceci cause une augmentation très rapide de la consommation d'énergie par rapport à l'énergie disponible dans les batteries. De plus, ce problème se retrouve avec les prises de courant qui fournissent une quantité fixe de puissance.

Il faut donc améliorer la consommation de puissance pour utiliser tout le silicium sans consommer trop d'énergie. Le processus traditionnel de développement utilise un Langage de Description Matériel (HDL) et des outils pour synthétiser le langage. Ces mêmes outils fournissent un logiciel qui permet d'estimer la consommation de puissance. Par contre, pour évaluer la consommation il faut synthétiser le circuit et le simuler pour connaître la consommation. Ce processus est très coûteux en temps de synthèse, de simulation et d'estimation de puissance. Car il est recommencé autant de fois que le résultat obtenu n'est pas satisfaisant.

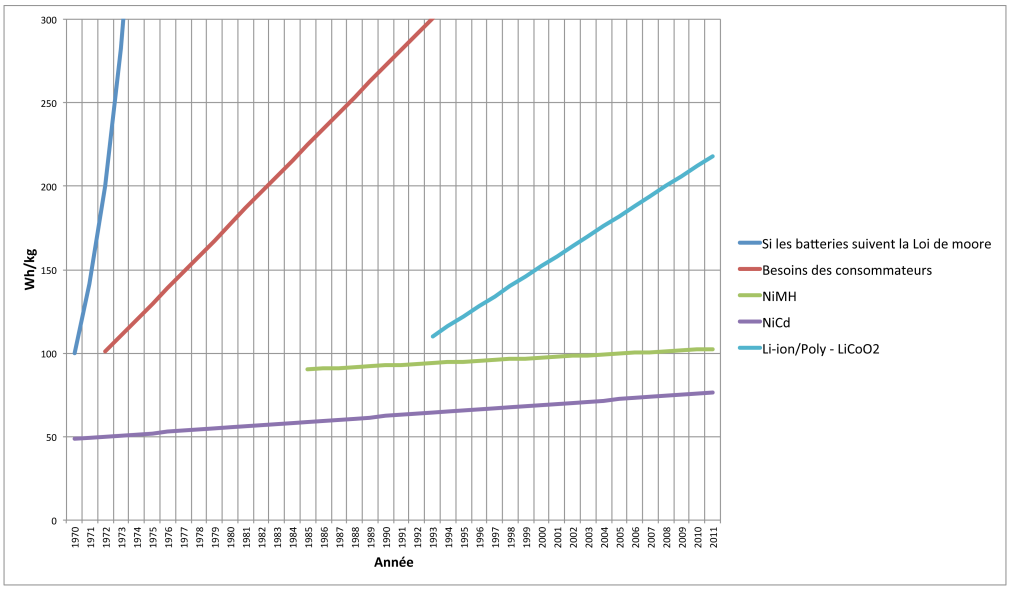
1.1 Problématique

La consommation de puissance est définie comme la quantité d'énergie consommée durant un certain temps. Puisque le domaine de l'électronique utilise des transistors, les modèles de consommation sont basés sur leur équation de puissance. Mais, les transistors dans les systèmes numériques sont utilisés dans leur mode interrupteur. La consommation est représentée par une partie statique et dynamique. La consommation statique est due au courant de fuite. La consommation statique peut-être déterminée en connaissant les ressources utilisées par le circuit et en connaissant le coût en consommation de chaque ressource.

La puissance dynamique est due à l'activité dans le circuit. La puissance dynamique se définit comme suit.



(a) Loi de Moore [30]



(b) Densité d'énergie [42]

Figure 1.1 Augmentation du nombre de transistor et de la densité d'énergie

$$P = ACV^2 f \tag{1.1}$$

où A correspond à la probabilité d'avoir une activité qui est appliquée à la capacité C avec un voltage V. f correspond à la fréquence de fonctionnement. Si l'activité n'est pas régulière, il est possible de changer Af par la valeur équivalente n/t où n correspond au nombre de fois que l'activité se passe durant un temps t.

$$P = \frac{nCV^2}{t} \tag{1.2}$$

L'équation 1.2 permet de déterminer la consommation de puissance de n'importe quel circuit analysé au niveau du transistor. Dans le cas d'une analyse de «Field Programmable Gate Array» (FPGA), on remplace les transistors par les cellules logiques. Ceci impose de modifier les valeurs C et A.

$$P = \frac{nA(\ldots)C(\ldots)V^2}{t} \tag{1.3}$$

La raison vient du fait que la charge de la cellule change dépendant de sa configuration et que l'activité peut venir de plusieurs ports. Bien que l'équation 1.3 permette maintenant de simplifier les modèles en analysant l'activité des cellules logiques, les FPGA ont une haute densité de cellules logiques dans les millions dans une puce [59]. Bien que l'on ait moins de composantes à analyser (une cellule logique comprend des centaines de transistors [59]), la complexité de l'équation 1.3 augmente, ce qui limite l'accélération de l'estimation.

À cause de la complexité des systèmes, le remplacement du HDL pour un langage plus adapté à la représentation haut-niveau devient une option intéressante pour accélérer le temps de développement et obtenir une estimation de consommation plus rapidement. SystemC offre une excellente performance de simulation et est supportée par plusieurs Environnement de Développement Intégré (IDE). Par contre, SystemC manque encore de support pour effectuer l'estimation de consommation de puissance.

L'avantage du SystemC au niveau Modélisation au Niveau Transactionnel (TLM) est de pouvoir faire abstraction des détails de l'implémentation comme la technologie (Circuit Intégré à Aplication Spécifique (ASIC) ou FPGA) et de permettre à l'utilisateur de se concentrer sur l'application. Cette particularité cause des problèmes à l'estimation de puissance, car il n'est pas possible de connaître la charge et la distribution de l'activité sur cette charge. Pour

contrer ce problème, il est possible de créer une librairie d'estimation qui tente de prédire le Placement et Routage (PAR) pour calculer la charge et l'activité appliquée à cette charge grâce au routage. Cette prédiction est difficile à réaliser à ce jour, car le système décrit sera converti en un langage HDL et synthétisé pour configurer un FPGA. Cette double conversion empêche d'estimer précisément la consommation en se basant sur des simulations du code SystemC et des prédictions de PAR.

Il existe une solution qui permet de tenir compte du niveau d'abstraction du langage SystemC pour permettre une estimation précise. Cette méthode utilise une abstraction similaire à celle utilisée par les estimateurs FPGA vis-à-vis des transistors. La méthode utilise les IP comme structure de base et simule le comportement de la composante. Il ne suffit que de porter les configurations à l'implémentation finale pour obtenir un comportement similaire qui sera affecté par le PAR. En considérant l'impact des configurations sur le PAR il est possible de faire une estimation de puissance. Cette méthode change le point de vue de l'équation pour déterminer la charge par rapport aux configurations de l'IP et de sa technologie d'implémentation. Ceci permet d'estimer des systèmes complexes avec un minimum d'équation. De plus, cette méthode permet de ne pas tenter d'estimer la partie du PAR.

1.2 Objectif

L'objectif de ce travail est de présenter une méthodologie pour créer des modèles de consommation de puissance dynamique pour des simulations TLM. Cette méthode devra réaliser des modèles pour différents IP qui auront différents paramètres architecturaux. Établir une méthodologie d'estimation de puissance est très ardu. Pour assister dans ce travail, un ensemble d'outils en Perl seront créés pour effectuer les mesures et la validation des résultats. Ces outils contrôleront les applications de Xilinx pour automatiser la synthèse et les mesures de référence de puissance. Il sera important que les modèles utilisent un minimum de paramètres pour minimiser l'impact sur une simulation.

1.3 Méthodologie

Il faut établir les paramètres à mesurer pour chacun des IP. Ceci consiste à sélectionner les paramètres communs à tous les IP d'une même catégorie (mémoire, processeur, etc...). Par la suite, il faut établir la relation entre ces paramètres et la consommation pour en extraire une équation qui représentera la consommation. Cette fonction sera construite à partir des options ou paramètres structurels de l'IP et de son activité. Le modèle de consommation sera contruit en suivant une méthodologie qui établit la strucutre des équations pour obtenir un modèle réutilisable. Les modèles de consommation seront testés avec un système, dont

l'architecture sera paramétrable tout comme l'application exécuté par le processeur. Les applications utilisées seront le Dhrystone [48] et le Coremark [13]. Les résultats seront comparés à un estimateur «Register Transfert Level» (RTL).

1.4 Contribution

Ce travail propose une méthode permettant d'obtenir un modèle de consommation de puissance d'un IP qui est à la fois simple et précis. Cette méthode utilise les particularités d'un IP pour diriger la formation de l'équation et cibler les paramètres pertinents à l'estimation. En permettant ce ciblage, le modèle résultant est à la fois plus simple par rapport à d'autres méthodes et permet une estimation précise. Cette méthode est aussi applicable à tous les IP pour lesquels on veut une estimation de consommation.

De plus, cette méthode permet de réutiliser les équations de modélisation. Que ce soit pour un IP de la même famille, pour des configurations différentes, une nouvelle architecture système ou des contraintes d'implémentation différentes. Le modèle utilise des paramètres qui sont communs à une famille d'IP (certains IP auront des paramètres uniques pour représenter des particularités uniques) pour pouvoir facilement créer un modèle de consommation. Les modèles sont basés sur la caractéristique structurelle de l'IP et son activité particulière (lecture, interruption, instruction, etc...). À cette équation des paramètres d'implémentations (fréquence, technologie) sont ajoutés pour cibler la consommation de l'implémentation réalisée.

Ce travail a mené à une publication d'article. Le projet a été présenté à la conférence ICCD 2010 [40] où la méthode a été démontrée pour le processeur et la mémoire.

1.5 Distribution des Chapitres

Le mémoire présente un survol de l'état de l'art de l'estimation de la consommation au Chapitre 2. Par la suite, la méthode explorée dans ce mémoire sera détaillée et discutée au Chapitre 3. La méthode sera appliquée sur différend IP comme la mémoire, le processeur, un bus, une minuterie et un gestionnaire d'interruption au Chapitre 4. Une discussion sur chacune des étapes sera faite pour expliquer les choix et résultats obtenus durant les mesures. Le Chapitre 5 analysera la performance des estimations de puissances par rapport aux décisions prises au chapitre précédent.

CHAPITRE 2

REVUE DE LITTÉRATURE

Déterminer la mesure de la consommation seulment une fois que le design est complété, n'est pas une option viable dans la majorité des projets, car cette méthode est itérative et longue. Alors, il faut trouver une solution qui permet de connaître la consommation de puissance au début ou durant le processus de développement. Plusieurs méthodes permettent de faire cette estimation en utilisant différentes solutions. L'estimation peut être effectuée à partir de modèles basés sur la technologie d'implémentation, se baser sur le comportement d'un circuit ou sur une analyse par IP (à plus haut niveau).

2.1 Estimation au niveau matériel

Une des premières méthodes d'estimation de puissance développée est l'estimation effectuée au niveau matériel (transistor ou «gate-level» et RTL). Ces méthodes d'estimations sont couramment utilisées encore aujourd'hui, car elles permettent une estimation très précise. De plus, les fournisseurs d'outils pour les FPGA et ASIC fournissent un excellent support pour effectuer ce type de simulation. La fabrication d'un masque pour produire des ASICS peut coûter des millions de dollars. Une erreur, tant dans le circuit que dans l'estimation de la consommation d'énergie du ASIC, doit être évitée à tout prix. Par contre, ces simulations prennent un temps énorme (dans l'ordre des heures de simulations pour obtenir une seconde).

Les ASICs ne sont pas la seule technologie qui a besoin de limiter sa consommation, il y a les FPGA. Bien qu'il soit plus facile de changer le circuit dans un FPGA, cette technologie consomme plus d'énergie et est souvent utilisée dans des projets avec des temps de développement plus court. L'estimation de la consommation de puissance devient donc importante pour permettre de répondre aux exigences de consommation dans certaines applications embarquées, source de courant limité et dissipation de chaleur.

2.1.1 Estimation au niveau transistor

Cette section présente les premières méthodes d'estimation de puissance qui sont couramment utilisés aujourd'hui. Ces méthodes se situent au niveau du transistor ou de la porte logique.

Une des premières méthodes d'estimation utilise le transistor. À leurs débuts, les transistors et le routage étaient faits à la main. Ceci permettait de faire des estimations de

consommation de puissance directement avec le modèle du transistor. En utilisant la simulation de très haute précision utilisée par les ASICs, il est possible d'obtenir une estimation très précise. Par contre, les ASICs sont rapidement devenus trop complexes pour estimer la consommation de puissance avec de simples équations de transistor. Ce phénomène s'est accéléré avec l'apparition des langages de description matériels (HDL comme le langage VHDL et Verilog) et les FPGA. Ces derniers permettent d'utiliser des milliards de transistors [59] facilement en les regroupant en cellules logiques identiques distribuées sur le silicium.

Pour cette raison, il a fallu recourir à des modèles ciblés à l'application du circuit [27]. Dans cet article, l'auteur cible les équations pour un type de circuit en particulier, et ainsi spécialise le modèle d'analyse d'un modèle générique. Ce dernier utilise des hypothèses pour ne conserver que les facteurs pertinents. Il est possible de définir des équations pour la logique combinatoire, les bascules et les verrous (latch), la mémoire embarquée, les interconnexions (tant entre les portes logiques que les bus locaux), les bus globaux, l'horloge et les entrées/sorties. Puisque chacun est formé de différentes composantes, connectées de différentes façons, ceci permet de simplifier l'équation générale qui régit les circuits numériques. Ainsi, seuls les paramètres utiles pour le circuit sont utilisés. Par contre, ce ne sont que des versions simplifiées des équations normalement utilisées par les simulateurs de type CAD. Alors, il faut effectuer la même simulation globale du système en temps continu (ce qui permet d'obtenir les micro-impulsions). Cette simulation est ensuite utilisée en conjonction avec le modèle mathématique. Le modèle est connu lors de l'implémentation du PAR dans la technologie ciblée, car il utilise les paramètres de la technologie ou des statistiques d'activités («fan-out», «fan-in», largeur moyenne de la base des transistors, etc...).

Bien que cette méthode permette de faire des estimations précises, elle est limitée par son grand niveau de détail, tant dans le modèle que dans la simulation à effectuer. Une méthode d'accélération de l'estimation est de changer les équations pour minimiser le niveau de détail de la simulation. Deux modèles offrent une solution à ce problème : le modèle probabiliste ou statistique [36]. Le modèle probabiliste propage les probabilités de transition au travers de la logique pour connaître l'activité du circuit. Le modèle statistique simule le circuit avec des vecteurs aléatoires dirigés (par l'usager) tout en analysant la consommation. Ces modèles permettent de simplifier les simulations en ne requérant que les transitions d'entrées car les modèles lient les entrées avec le reste des transitions dans le circuit. Avec ces méthodes, il est possible d'effectuer l'estimation de différents sous-circuits.

Les sous-circuits utilisés pour les modèles probabilistes ou statistiques sont les circuits séquentiels ou circuits combinatoires. Ensemble, ces deux sous-circuits sont capables de représenter la majorité des circuits. La méthode probabiliste tend à vouloir modéliser les circuits en une équation exacte. Dans le cas des bascules d'une Machine à États Finis (FSM), les com-

binaisons deviennent rapidement trop nombreuses pour facilement représenter exactement la probabilité de chaque changement d'état. Pour faciliter l'analyse, des approximations sur l'équiprobabilité de chacun des état ou l'indépendance des entrées de la machine à état sont effectuées. Dans le cas des circuits combinatoires, qui peuvent paraître plus simples à analyser, il suffit de tenir compte des portes logiques à deux entrées mises en cascade. De plus, il n'y a habituellement pas de boucles de rétroaction dans la logique combinatoire sans que le circuit devient séquentiel. À cause de ces caractéristiques, il est effectivement plus facile de calculer le modèle probabiliste d'un circuit combinatoire si l'on ne tient pas compte des délais. Les délais causent les micro-impulsions qui sont une cause importante de la consommation d'énergie dans un circuit combinatoire. Une méthode proposée permet d'étendre l'estimation probabiliste pour résoudre la problématique de la corrélation spatiale et temporelle [47]. Cette méthode permet d'estimer la consommation avec une plus grande précision tout en offrant une accélération sur les techniques utilisés auparavant. La méthode statistique offre une flexibilité par rapport au circuit que l'on analyse, en masquant les détails de l'activité, au dépend d'une simulation plus longue en temps.

La méthode statistique offre la possibilité de soumettre un circuit sans tenir compte des détails internes au circuit. Pour cela, il faut lui fournir des vecteurs aléatoires dirigés pour maximiser le nombre et la couverture des transitions dans le circuit. L'avantage indéniable est de pouvoir choisir une précision ou de garantir un résultat avec une certaine précision. Si on analyse un circuit combinatoire, le modèle va tendre vers la moyenne de la consommation du circuit. Pour les bascules d'une machine à état, une analyse avec une méthode à la Monte-Carlo [1] est utilisée. Malgré cela, si le circuit comporte plusieurs branchements, il est possible de passer énormément de temps avant d'avoir un résultat stable et fiable. De plus, il faut s'assurer que le générateur de vecteurs aléatoires soit le plus aléatoire possible. Dans le cas des circuits combinatoires, cette méthode tend à donner une valeur moyenne de la consommation du circuit ce qui peut être non désiré si le circuit a un comportement très varié. Il faut alors passer à une analyse statistique au niveau de la porte logique, ce qui peut être très long si le circuit contient plusieurs portes logiques.

Pour faciliter l'analyse de la consommation au niveau logique, il faut distinguer les circuits séquentielles et les circuits combinatoires. On suppose que l'information sur l'activité

1. La méthode Monte-Carlo utilise des entrées aléatoires qui, sélectionnées par rapport à l'équation, permettent de connaître une approximation de cette équation. Plus on utilise de vecteurs aléatoires et que ces vecteurs aléatoires sont bien générés, plus le résultat sera précis. Si l'on prend un point $M(x, y)$ où $0 < x < 1$ et $0 < y < 1$ et que la valeur de x et y répond à cette équation $x^2 + y^2 < 1$, alors M fait partie du cercle de rayon 1 centré en $(0, 0)$. La probabilité d'avoir un point dans ce cercle est de $2\pi * r^2/4(x * y) = \pi/4$. Le 4 provient du quart de cercle qui se superpose au carré. Les vartiable x, y, r ont une valeur de 1. En calculant la proportion de points M dans le cercle, il est possible de connaître une approximation de π. Plus on augmente le nombre de points, plus la précision augmente. Il faut pour cela avoir un bon algorithme aléatoire [50].

des entrées soit connue [35]. Pour un circuit séquentiel, l'information peut s'obtenir de façon statique (seulement si les entrées sont connues). Dans le cas d'un circuit combinatoire, l'information doit absolument s'obtenir à partir d'une simulation ou à partir d'un calcul de probabilité (ceci s'applique aussi pour un circuit séquentiel dont l'état des entrées n'est pas connues). Si on veut minimiser le nombre de simulations et leur grosseur, il est possible d'effectuer une estimation par macromodel au niveau RTL.

2.1.2 Estimation au niveau RTL

L'estimation au niveau RTL se base fortement sur l'estimation de sous-circuits connu pour accélérer l'estimation de puissance. Trois types de modèles sont utilisés pour conserver l'information : l'équation, la table de valeurs et la table d'équations [3].

Le modèle par équation est plus connu. Il suffit de commencer par sélectionner les paramètres définissant l'activité du circuit. Il faut par la suite faire des mesures de consommation avec des combinaisons de paramètres incrémentés à intervalle régulier. Ensuite, il suffit de diminuer l'intervalle aux régions qui contiennent des variations de consommation plus grande. Il est possible d'éliminer certains paramètres si on se rend compte du faible impact de certains d'entre eux sur la consommation. L'avantage de cette méthode est sa flexibilité, la possibilité de faire de l'interpolation facilement et la rapidité avec laquelle il est possible d'obtenir un modèle. Mais, bien qu'il soit possible de faire de l'extrapolation, il faut faire attention au résultat obtenu. Lorsque l'on sort de la page de mesure effectuée, les résultats peuvent diverger puisque le modèle n'a pas été validé. Pour augmenter la robustesse des modèles, il est possible d'utiliser un modèle par table de valeur.

La méthode par table de valeur (3Dtab) estime al consommation de puissance selon trois paramètres : P_{in}, D_{in} et D_{out}. P_{in} définit la probabilité moyenne d'avoir une certaine entrée (une valeur précise ou un motif précis). D_{in} est la densité moyenne de transition d'entré et D_{out} est la densité moyenne de transition de sortie. La densité de transition est définie comme la probabilité de transition par unité de temps [34]. Chacune des trois dimension est divisée en intervalle égal de N point. Il est à noter, que P_{in} et D_{in} ne sont pas entièrement indépendants ; il est possible d'utiliser l'équation $D_{in}/2 \leq 1 - 2 * |P_{in} - 0.5|$ pour éliminer certains points et simplifier la table. Il faut analyser chaque point (P_{in}, D_{in}) et extraire un vecteur d'entré qui respecte les caractéristiques du point (P_{in}, D_{in}) dans sa globalité. La simulation produit une valeur D_{out} qui sera associée à la valeur de consommation de puissance mesurée (ce qui forme une table 3D en utilisant les trois paramètres). Certaines valeurs D_{out} peuvent ne pas avoir été couvertes une fois que toutes les combinaisons de (P_{in}, D_{in}) ont été analysées. Ceci est normal, puisque D_{out} est assujetti au comportement du circuit. Il est possible de trouver des patrons d'entrées valide pour un (P_{in}, D_{in}) donné qui permet de couvrir les D_{out} manquant.

S'il n'est pas possible de trouver un motif, il faut faire des interpolations ou extrapolations pour combler les trous. Toute cette procédure demande un grand temps de caractérisation qui tend vers $O(\frac{N^3}{2})$.

La méthode par tableau d'équation (EqTab) utilise la même prémisse que la 3DTab, par contre chacune des paire (P_{in}, D_{in}) est considérée comme indépendante. La raison est que la sortie D_{out} est définie par une équation dépendant de (P_{in}, D_{in}). Ceci fait tendre le temps de calcul vers $O(\frac{N^2}{2})$ ce qui est un ordre plus rapide que le calcul du modèle 3DTab.

L'utilisation de ces méthodes est limitée par le fait qu'elles représentent un circuit RTL très simple et leur génération peut prendre de une minute à huit heures. Bien que dans plusieurs cas le résultat soit rapide à obtenir, il est peu pratique de simuler un petit circuit RTL pendant 8 h. Ces méthodes demandent de choisir entre deux limitations. On peut choisir d'effectuer une simulation très longue pour couvrir tous les cas de figure des modèles par tables. Sinon, il faut accepter une erreur additionnelle due à l'interpolation et l'extrapolation. Ces choix ne rendent pas les méthodes d'estimation RTL très intéressantes.

2.1.3 Émulation de Puissance

Une façon d'accélérer la vitesse de l'estimation de puissance RTL est de pouvoir utiliser l'implémentation RTL obtenue et de l'exécuter sur un FPGA. Puisque les estimations RTL sont si lentes, il suffit de faire l'estimation sur le FPGA qui sera capable de faire avancer la «simulation» à la vitesse réelle d'exécution [12]. Pour faire une estimation rapide de la consommation de puissance, il faut implémenter dans le FPGA le modèle de puissance du circuit qui fait l'analyse de la consommation durant son exécution. En se branchant à des interconnexions, il est possible d'obtenir des statistiques d'activités du circuit et calculer au fur et à mesure la consommation du circuit. Cette méthode ne fait qu'accélérer la vitesse de modélisation du circuit car elle reprend des modèles synthétisés pour les mettre en forme matérielle. Donc, il faut toujours passer par la synthèse longue car il faut maintenant ajouter le modèle de puissance synthétisable. De plus, le modèle augmente l'aire requit de 3 à 4 fois pour pouvoir analyser le circuit, même après avoir eu recours à plusieurs techniques d'optimisation de ressource. Avec un circuit qui utilise 20% du FPGA on obtient une utilisation finale de 60 - 80%. Ceci limite les possibilités d'analyse de circuit qui requiert plus de 33% des ressource disponible avant l'ajout des modèles. Pour ne pas influencer l'analyse, il faut que l'usager perturbe le PAR du circuit le moins possible lorsqu'il ajoute les modèles. Le PAR choisi sera celui généré par le synthétiseur pour le circuit seul dans le contexte de son utilisation prévue. Ensuite, il faudra ajouter le circuit d'analyse et des interconnexions entre le système et le circuit d'analyse. Si le circuit a des contraintes de performance en fréquence ou un PAR avec une partance élevée, il sera très difficile d'ajouter des interconnexions qui traverseront

le Circuit sous Test (DUT). Ceux-ci seront de surcroît très parasitique. Tout ceci risque de causer des problèmes durant l'analyse et des imprécisions dans les résultats.

Dans le cas d'un circuit à très haute non-linéarité, comme un processeur ou une machine décisionnelle quelconque, ce type d'analyse perd rapidement ses possibilités de minimisation. Il faut tenir compte de cas très complexe comme les «cache miss», la gestion d'erreurs et prédiction de branchement. Pour cela, il faut trouver une possibilité d'augmenter la flexibilité du modèle accéléré matériellement en conservant son gain de vitesse.

Une méthode cible ce problème en utilisant une simulation hybride logiciel-matériel [17]. L'idée est de décharger la simulation logicielle sur le FPGA pour accélérer la simulation des modèles trop complexes. Dans le cas d'un modèle de puissance trop gros, il est possible de mettre le modèle d'analyse en logiciel et le module fonctionnel du circuit sur le FPGA. Ceci permet d'avoir une simulation matérielle et une exécution du modèle de puissance très rapide malgré leur complexités. Il est donc possible d'avoir un juste équilibre entre rapidité et flexibilité qui répond au besoins de l'utilisateur et aux limitations du FPGA. Pour certains cas, les signaux du système de mesure ne peuvent pas être ajoutés même si le modèle de puissance est relégué sur l'ordinateur, dans le cas d'un modèle ayant des contraintes très serrées. Le problème majeur de cette méthode est sa complexité d'implémentation. Il faut s'assurer de la synchronisation des données des modèles et de la valeur de la consommation dans le temps. Les modèles de puissance sont implémentés comme des pipelines qui donnent un résultat de consommation à chaque cycle. Puisque le délai de chacun des modèles peut varier, il faut resynchroniser les différentes valeurs. Un autre problème survient lorsqu'on essaie d'extraire l'information. Les modèles d'analyse génèrent un grand nombre d'informations, pour cela il faut avoir une interface disponible qui est capable de fournir la bande passante.

Les deux méthodes ne permettent pas de faire une analyse en début de développement car il faut toujours un modèle RTL synthétisé pour en faire l'analyse. Bien qu'ils rendent leur utilisation plus réaliste, il est peu probable que leur résultat arrive assez tôt dans le projet pour amener des corrections qui ne retarderont pas le projet. Les analyses et les budgets de puissance sont faits au début du projet. À la fin, les analyses ne sont faites que pour confirmer que les budgets ont été respectés.

Une solution tente de répondre au problème d'ajout d'un module de surveillance dans un circuit. Cette solution utilise les bus qui sont implémentés pour interconnecter les microprocesseurs et les mémoires pour brancher une Unité de gestion de puissance (PMU) au bus [10]. Ceci lui permet de suivre la consommation du processeur et de ses accès mémoires. Si le processeur vient avec un PMU, il le réutilise. Le PMU suit chaque instruction du microprocesseur et les transactions aux mémoires. Ainsi, l'intrusion du circuit de calcul de consommation de puissance est gardée au minimum. Par contre, le PMU ne tient pas compte des transactions

effectué ni de toute l'activité dans le circuit. En utilisant cette méthode de suivi d'exécution, il serait plus profitable de monter le niveau de représentation à une simulation précise au cycle avec un langage comme SystemC. Ce type de simulation permet de faire l'analyse plus tôt dans le processus de design et permet d'avoir autant, sinon plus, d'information que peut obtenir le PMU.

2.2 Estimation Haut-Niveau

L'estimation à haut-niveau tente de résoudre des problèmes inhérents à la modélisation RTL, qui est l'obligation de faire une synthèse et la lenteur de la simulation. Par contre, il ne suffit pas de simplement changer le langage. Il faut aussi adapter la simulation pour profiter des avantages de la modélisation à haut niveau.

2.2.1 Estimation haut-niveau avec granularité au transistor

Le fait d'utiliser un langage différent que ceux utilisés en RTL (p. ex. VHDL ou Verilog) ne permet pas d'effectuer une simulation à haut-niveau en elle-même. Par exemple, exécuter un modèle de circuit avec une librairie de cellule standardisé en C et Matlab ne résoud pas nécessairement les défauts d'une analyse RTL traditionnelle [26]. La méthode observe l'effet des transitions des entrées et des sorties sur la consommation de chacune des entités de la librairie de cellule. La base de cette méthode simule les signaux d'une façon différente (transition utile au lieu des transitions et micro-impulsions des signaux), le fait d'analyser chaque élément de cette façon pour un ASIC rend le système plus rapide que la méthode RTL. Dans ce cas, il est possible d'accélérer l'estimation de puissance d'environ 250 fois pour un multiplieur à 4 bits. Le temps nécessaire pour l'analyse est de 52 secondes, ce qui est long pour l'estimation d'un multiplieur. Pour des systèmes embarqués avec des multiplieurs de 32 bits on peut monter à une simulation de plus d'une heure, et cela sans même l'ajout du reste du circuit. Aussi, cette méthode ne tient pas compte de l'effet de la synthèse et du PAR sur la consommation de puissance. Si l'on effectue le même processus pour un FPGA le problème reste le même [11, 21]. Le temps nécessaire à modéliser les effets des délais sur les transition des signaux est énorme. Cela annule la simplification de la simulation amenée par les modèles à hauts-niveauxsans effectuert pas une abstraction des delta cycles de simulation.

2.2.2 SystemC

SystemC n'est pas une méthode en soi. Ce langage permet d'élever le niveau de la simulation dans un langage plus flexible et une philosophie de design différent du RTL. SystemC

permet de faire des analyses de circuit en analysant son comportement plutôt que son architecture. Un bon exemple de l'utilisation de SystemC comme plateforme d'estimation de puissance est PowerSC [24]. Cette plateforme permet de faire l'analyse de puissance d'un design en utilisant un ensemble de macromodèles (plus de détail sur ces modèles sera donné dans la Section 2.2.2.1). PowerSC requiert un modèle comportemental pour chacune des composantes et des macromodèles correspondants (4 pour chaque module). Une fois que les modèles sont créés, il faut ensuite reprendre ces modèles et les mettre dans une librairie qui sera ajoutée au SystemC pour faire l'analyse de puissance. Pour chaque analyse, PowerSC a le choix de prendre la méthode d'estimation avec ou sans corrélation entre les macromodèles. Si PowerSC utilise le modèle sans corrélation, il sélectionne le meilleur modèle de la composante dans la librairie pour le cas particulier d'activité. Par contre, si le modèle avec corrélation est sélectionné, PowerSC module l'apport à la consommation de chacun des modèles par rapport à leur corrélation (correspondance) avec le patron d'activité.

PowerSC utilise quatre modèles pour effectuer ses estimations de consommation. Ceci lui permet d'avoir une meilleure précision en sélectionnant le modèle le plus adapté pour chaque circuit. Par contre, la librairie augmente en complexité puisqu'il faut maintenir quatre modèles et ajouter une infrastructure de sélection des modèles et effectuer l'estimation.

En effectuant une estimation d'un circuit décrit dans un langage comme SystemC de la même façon qu'un langage HDL, on ajoute une grande incertitude vis-à-vis de l'estimation de la consommation à cause des outils de synthèse C vers le RTL comme Catapult C de Mentor [28] ou Forte Cynthesizer [15]. Le résultat de la synthèse SystemC vers le RTL doit par la suite passer au travers d'un synthétiseur RTL. Cette double interprétation du code C cause de grandes variations dans les résultats d'un même circuit dans un contexte (contraintes) différent ou en présence d'autres circuits. Malgré la création d'un bon modèle pour le SystemC, les optimisations et PAR des synthétiseurs causeront beaucoup de différences avec le modèle «idéal». L'effort investi à analyser les circuits, comme des circuits RTL, cause plus de problèmes. À cause des multiples outils de synthèse utilisés qui fournissent des résultats qui peuvent être affectés par la façon de décrire le circuit, il faudrait pouvoir prédire les résultats des synthétiseurs. En utilisant la synthèse à haut niveau, il est possible de définir l'implémentation d'une mémoire dans un FPGA [38] à partir de la description en C pur et des contraintes du système. En étendant la synthèse haut-niveau à un système complet, il sera possible d'utiliser PowerSC. Plusieurs méthodes partielles de synthèse haut-niveau [16] sont proposées et permettent de synthétiser différents types de systèmes (par exemple un MPSoC) ou des parties d'un système.

2.2.2.1 Macro-Modélisation

L'utilisation de macromodèle a été présenté à la Section 2.2.2. Les macromodèles comme leur nom l'indique sont des modèles qui réfèrent à un ensemble de sous-circuits ou un groupe de circuit facilement modélisable par son comportement prédictible. Un exemple de ce dernier cas est montré dans l'article [1] où la prédictibilité d'un circuit est utilisée pour définir un modèle compact et précis. Les exemples de circuits analysés sont la Transformée de Fourier Rapide (FFT), le décodeur viterbi [29] et le processeur ARM. Dans les deux premiers circuits, il est possible de représenter le comportement par une machine d'état. Puisque l'analyse est faite sur la consommation moyenne basée sur les transferts effectués avec le protocole Wimax [49], le modèle se simplifie en analysant uniquement l'activité pertinente. Pour le processeur, ils représentent l'état de fonctionnement du processeur au lieu du calcul. Puisque le processeur effectue le même traitement en boucle, il est possible de représenter chaque état par une consommation moyenne stable. Il suffit d'ajouter la transition entre les états pour obtenir une représentation similaire à la FFT ou au décodeur Viterbi. Les circuits analysés sont très réguliers dans leur traitement et comportement, ou, comme dans le cas du processeur, qui tend vers une consommation moyenne pour chacun des états de traitements. Ici, la particularité du système est la régularité du traitement et de l'activité dans les circuits. Sans cette particularité, le circuit ne serait pas facilement représentable par une machine d'état. Cette méthode est limitée pour une situation où le comportement du circuit peut-être très varié dans le temps. L'élaboration de la machine d'état de toutes les possibilités peut devenir impossible (p. ex. un processeur exécutant un logiciel de contrôle avec interruption).

L'analyse par macromodèle d'un processeur est très complexe à cause du grand nombre d'état. Pour cela, il faut changer le point de vue du processeur vers le logiciel [22]. Pour analyser le logiciel, il est possible de faire abstraction du processeur pour analyser le graphe de flot de contrôle du programme. Le programme est subdivisé en blocs qui représentent un ensemble d'instructions qui sont toujours exécutées ensemble. Il utilise la corrélation (la probabilité d'exécution d'une branche en relation avec la branche précédente) pour augmenter l'analyse du logiciel. Deux méthodes différentes sont utilisées : la première compte l'exécution d'un nombre fixe de blocs de code et la seconde compte le nombre de fois qu'une série de blocs de code de longueur variable est exécutée. La série de blocs variable est définie entre le point de départ, ou le bloc pointé par un retour en arrière (un lien logique d'un graphe qui crée une boucle), et le point de sortie du logiciel, ou la source du retour en arrière. Il suffit de faire le décompte du nombre d'exécutions de chacune des corrélations de bloc et de le multiplier avec le bon coefficient pour en connaître la consommation d'énergie. Le choix entre les deux dépend surtout de l'accélération de l'estimation voulue et de la précision recherchée. Cette méthode devient directement liée au logiciel exécuté puisque l'on analyse le programme par

rapport aux corrélations de ses blocs de code. De plus, pour un autre processeur, il faudra recommencer l'analyse puisqu'il n'y a aucune façon de lier l'équation au matériel disponible (instructions, registres, etc...) pour exécuter le code.

2.2.2.2 Extension en Modélisation par IP

Pour pouvoir faciliter l'analyse de systèmes plus complexes composés de processeur, mémoire ou bus, il faut utiliser une méthode plus globale qui est capable de cerner les similitudes entre les circuits de même fonctions. Ceci permet d'utiliser le comportement du circuit et sa spécificité à notre avantage pour simplifier le modèle d'analyse de consommation de puissance. Un exemple de cette méthode est démontré dans [14]. Dans cet article, les auteurs analysent la consommation d'une mémoire BRAM de Xilinx en se basant sur les paramètres qui définissent l'IP et son fonctionnement. Pour analyser la mémoire, ils utilisent un modèle qui a besoin du nombre de bits d'adresse et de donnée, le nombre de bits en transition et de la fréquence. Leur modèle se divise en trois parties. La première partie consiste à déterminer l'impact de l'activité de l'adresse sur la consommation globale, et la seconde partie consiste à déterminer l'impact de l'activité de la donnée sur la consommation globale. La dernière partie consiste à déterminer la consommation causée par la grosseur de la mémoire basée sur le nombre de bits de donnée et d'adresse (sans considération d'activité). Cette dernière partie détaille la consommation statique de la mémoire et la consommation de l'arbre de l'horloge. Le modèle utilise une régression linéaire pour définir l'équation et considère la lecture et l'écriture comme équivalente en terme de consommation. Le fait d'utiliser uniquement une régression linéaire limite la possibilité de prendre en compte l'effet exponentiel de la largeur de l'adresse sur la consommation. En effet en ajoutant un bit à l'adresse on double le nombre de cellules mémoires adressées. Du même fait, l'ajout d'un bit de donnée à de moins en moins d'effet. À chaque bit de donné, on ajoute $\frac{1}{n+1}$ transition de plus (si tous les bits transitent) où n est le nombre de bit. Une autre source d'erreur est d'approximer que l'écriture et la lecture consomment la même quantité d'énergie. Bien que les deux accès contrôlent un nombre équivalent de multiplexeurs/démultiplexeur, l'écriture charge l'entrée des bascules contrairement à la lecture. Pour minimiser l'erreur d'un modèle dérivé d'une régression linéaire, il faut procéder par une méthode automatique qui génère un modèle et ajoute des paramètres à ce modèle pour atteindre un niveau acceptable de précision en un temps raisonnable.

Une méthode de génération automatique de modèle de consommation de puissance est présentée dans [6]. L'article analyse l'IP matériel en le comparant à un modèle SystemC équivalent. La méthode commence par simuler le modèle SystemC en conservant l'activité sur les I/O et toutes les variables qui peuvent être utilisées pour calculer la consommation de puissance. Par la suite, en utilisant le même banc d'essai, ou vecteur de stimulation, le système

calcule la consommation de puissance fournie par un outil de consommation au niveau RTL (ou porte logique). La méthode sélectionne ensuite les paramètres pertinents au modèle en leur associant un coefficient de corrélation. Ce coefficient définit la probabilité que la variation de la consommation de puissance ne soit pas due à la variation de la variable. Une fois que la variable est définie comme statistiquement significative [45] le paramètre est conservé pour le modèle. Les variables avec un nombre limité de valeurs possibles qui ont un impact majeur sur la consommation (p. ex. écriture vs lecture) sont considérées séparément comme méthode de sélection d'équations pour la situation (qui utilise les adresses, les données, la fréquence, etc. . .). Par la suite, un modèle est défini par régression pour atteindre un niveau de précision voulu. Cette méthode permet d'avoir un modèle généré automatiquement. Par contre, une limitation apparaît quand, pour un contrôleur d'interruption, le nombre de variables analysées est un nombre considérable de paramètres (80 ou 160 si l'on considère la valeur précédente dans le temps). En étant automatisé, l'usager ne peut diriger le modèle pour simplifier le nombre de variable dans le modèle. La charge de calcul du modèle [45] sur le temps de la simulation est de 10% par rapport à la simulation sans modèle. La méthode est contrainte par la création d'un modèle SystemC équivalent en terme de structure pour en faire l'analyse. Le modèle n'est pas réutilisable, car il est généré avec une énorme quantité de paramètres sans que l'usager puisse diriger la forme finale de l'équation.

2.3 Présentation de la Méthodologie

Plusieurs méthodes qui estime la consommation de puissance ont été présentées. Par contre, elles contiennent plusieurs limitations soit dans leur rapidité ou des contraintes dans leur utilisation. Certaines de ces méthodes nécessitent un temps de simulation très long et dépendent d'une synthèse. L'émulation de puissance est limitée à des circuits relativement petit à cause de l'utilisation de ressource par les modèles d'estiamtion de puissance. Si l'on tente de diminuer les ressources, on se retrouve à distribuer l'estimation entre le FPGA et le simulateur, ce qui complexifie l'infrastructure en plus de diminuer l'accélération apportée par la méthode. Les méthodes à haut-niveau utilisent des modèles qui sont utilisables pour une seule application. Sinon, les modèles obtenus sont si complexes que les simulations sont ralenties par un facteur non négligeable.

Une méthodologie permet de concevoir un modèle de consommation de puissance à la fois simple, précis et réutilise les IP disponible, dans le but de faciliter l'exploration architecturale d'un design est proposée. Le modèle utilise des simulations TLM qui permettent à la fois d'obtenir des simulations rapides et d'obtenir les données pour calculer la consommation de puissance. Le modèle se subdivise en trois parties. Chacune de ces parties permet de modéliser

l'effet et l'apport de différents paramètres ou particularités architecturales du circuit. Un modèle peut se subdiviser en son comportement, ses particularités architecturales et son implémentation dans une technologie. Ceci permet de réutiliser les équations pour représenter un maximum de situations, implémentations et circuits avec un minimum de complexité.

CHAPITRE 3

DÉTAILS DE LA MÉTHODOLOGIE

L'intérêt de la méthodologie est de pouvoir abstraire la consommation de puissance d'un IP en équation simple pour une analyse au niveau TLM. Tel que décrit dans le Chapitre 2, il est très complexe d'analyser tous les paramètres influençant la consommation de puissance. Alors, la solution est de regrouper les paramètres qui influencent la consommation en différentes catégories qui reflètent leur effet et leur comportement. Il faut regrouper les paramètres d'une façon judicieuse pour ne pas complexifier les équations ou avoir un modèle inexact. De plus, il faut faire attention de bien distinguer l'effet des entrées et sorties des IP sur la consommation.

La méthodologie proposée est illustrée à la Figure 3.1 et elle est composée de 5 étapes. La première étape est la sélection et la catégorisation des Paramètres Structurel Actif (ASP), Paramètres Structurel Statique (SSP) et Paramètres Technologique (TP). La seconde étape consiste à évaluer l'impact des paramètres ASP et SSP qui est suivi dans la troisième étape de l'élaboration de l'équation correspondante. La quatrième est l'évaluation de l'impact des paramètres SSP et TP au cours de l'implémentation sur le modèle et permettra, par la suite, la cinquième étape élabore l'équation finale. Chacune des étapes sera appliquées sur des IPs utilisés dans un système sur puce (p. ex. processeur, mémoire, bus, etc.). Dans ce chapitre, la procédure générale sera détaillée pour donner une vue d'ensemble de la méthodologie sans tenir compte d'un IP en particulier.

3.1 Sélection et Catégorisation des Paramètres

La première étape consiste à énumérer les différents paramètres qui conditionnent la consommation de puissance d'un IP (l'adresse, la donnée, le type d'accès, la grosseur de la mémoire, l'instruction, le nombre de port, la fréquence et la technologie). En détaillant les paramètres, il sera possible de définir la consommation avec trois caractéristiques : 1) son activité, 2) sa structure et 3) son implémentation. Ceci permet de créer une structure qui sera commune aux IPs de même classe de fonctionnement. En définissant la consommation de cette façon, il est plus facile d'intégrer le modèle dans un simulateur TLM et d'interchanger les IP pour connaître leur effet sur la consommation globale. Parmi les paramètres sélectionnés, certains ont très peu d'effet sur la consommation du modèle. Il se peut qu'un IP utilise un paramètre que les autres n'ont pas (soit à cause de la structure ou de l'activité).

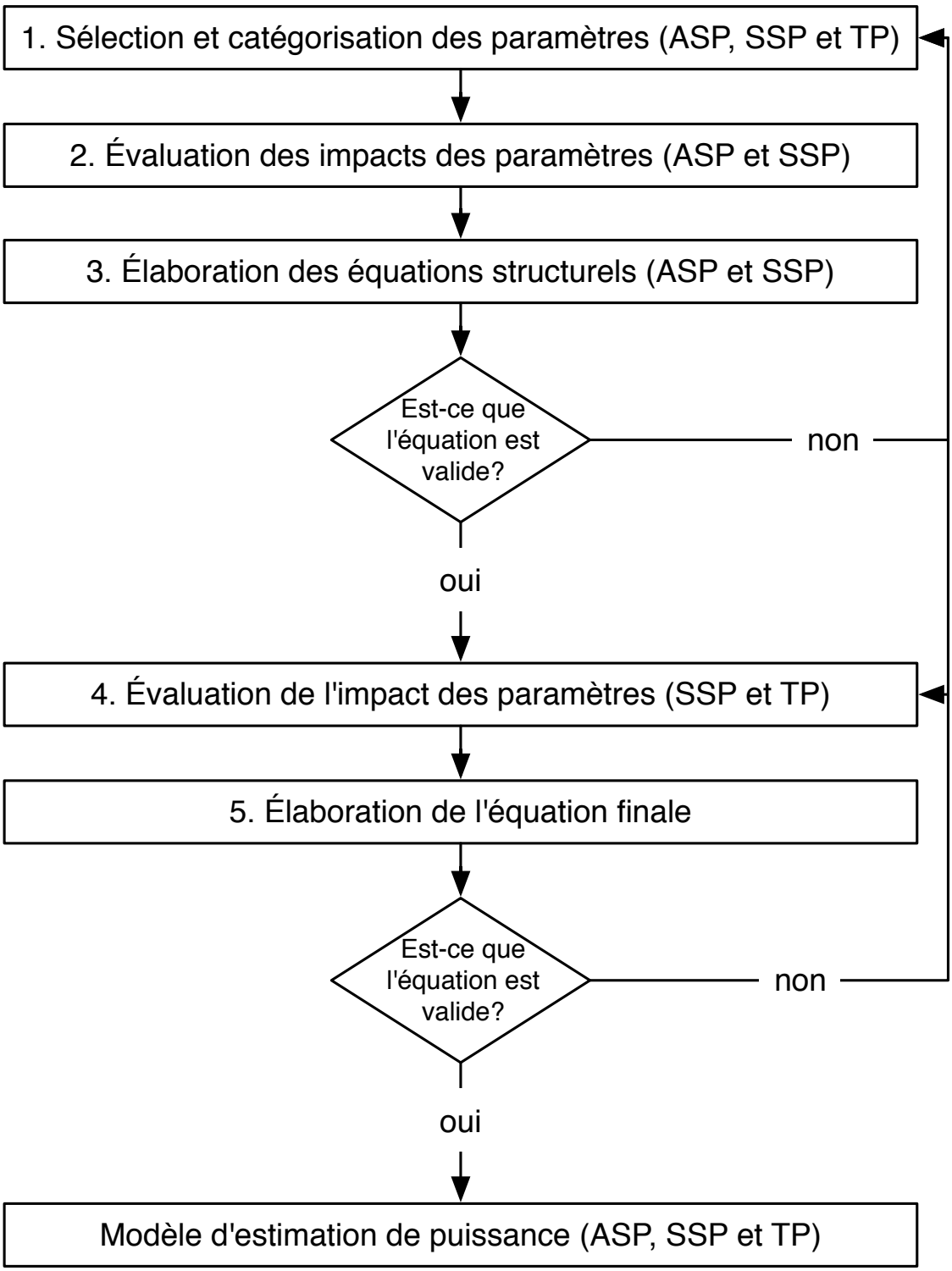


Figure 3.1 Flot de la méthodologie pour caractériser un IP

La première catégorie, ASP, regroupe les paramètres qui varient durant la simulation. Ceci peut être un état interne de l'IP, une action sur une entrée ou sortie, ou toute autre information pertinente qui fluctue dans le temps (p. ex. la donnée transmit, l'instruction exécutée, etc.). La seconde catégorie regroupe les paramètres structurels qui définissent l'architecture de l'IP (grosseur de la mémoire, nombre de modules, etc.). Au cours de la simulation, ce sont des valeurs qui restent stables et qui modulent l'impact des ASPs sur la consommation de puissance de l'IP. La dernière catégorie, TP, rassemble les paramètres qui influencent la consommation lors de son implémentation (fréquence de l'horloge, technologie et placement, et routage).

Les paramètres ASP et SSP forment le coeur du comportement de puissance de l'IP. Les paramètres TP affectent ce comportement en ajoutant l'influence de l'implémentation de l'IP. L'effet des TP s'ajoute au modèle par multiplication pour ajuster la consommation de puissance en relation à l'implémentation. La forme générale de cette relation est donnée par l'équation (3.1). Les paramètres SSP sont analysés avec les TP car l'impact de la technologie et de l'implémentation dépend aussi de la structure du système (SSP) qui influence la quantité de routage et de ressource utilisée.

$$M(TP, ASP, SSP) = \frac{\sum_{k=1}^{\text{nbr d'IP}} \left(F_k(TP, SSP) \sum_i \left(G_{ik}(SSP, ASP) \right) \right)}{\text{Temps de Simulation}(s)} \tag{3.1}$$

où $M(\ldots)$ correspond à la consommation de puissance totale, $F_k(\ldots)$ est la fonction de l'implémentation et $G_{ik}(\ldots)$ est la fonction détaillant l'effet de la structure et de l'activité sur la consommation. L'indice k représente les IP à analyser, tandis que l'indice i correspond à chaque stimulus, ou activité de l'IP, à travers le temps. Puisque les équations $F_k(\ldots)$ et $G_{ik}(\ldots)$ expriment une quantité d'énergie, il faut diviser le résultat par le temps de simulation observé.

Cette formulation de la consommation permet de réutiliser $G_{ik}(\ldots)$ pour différentes implémentations ou technologie puisque $F_k(\ldots)$ est l'équation qui est affectée par l'implémentation. La relation entre les différents paramètres (ASP, SSP et TP) sera détaillée dans les prochaines sections.

3.2 Évaluation de l'Impact des Paramètres

Une fois que les paramètres importants de l'IP sont définis et catégorisés, un processus automatisé peut-être utilisé pour stimuler chacun des paramètres et ainsi connaître son impact sur la consommation de puissance de l'IP. Puisque les paramètres qui font partie intégrante de la structure de l'IP, ou de son activité, sont connus, il sera possible de déterminer la forme

générale de l'équation qui approximera le mieux la consommation de puissance.

Dans la seconde étape, les paramètres ASP et SSP sont fixés à des valeurs de départs. Par la suite, chacun des ASP varie pour pouvoir couvrir tous les cas ou un échantillon représentatif des cas possibles. Ceci permet de couvrir les différentes entrées et sorties pour déterminer leur impact sur la consommation. Généralement, les IP sont composés d'entrée et sortie (I/O) qui ont différents impacts sur l'activité. Par exemple, la mémoire est affectée différemment par un changement de l'adresse, de la donnée ou des signaux de contrôle. Les entrées ou les sorties ne sont pas les seules à affecter la consommation. Par la suite, la couverture des paramètres ASP est refaite pour différentes valeurs de SSP qui couvre le spectre des options structurelles. Idéalement, le nombre de mesures est maintenu au minimum requis pour pouvoir construire un modèle précis. Un plan d'expérience peut-être utilisé initialement pour étudier la relation des différents paramètres. Si les effets des paramètres ne sont pas linéaires, il faudra procéder à des mesures supplémentaires pour pouvoir représenter le comportement. Ceci a pour effet supplémentaire de minimiser les effets parasites, comme les non-linéarités négligeables et de conserver le modèle relativement simple. En se fiant trop aux mesures sans extraire la tendance amenée par les paramètres, les équations augmenteront en complexité surtout s'il y a de la variabilité dans les résultats. Dans une optique d'estimation de puissance à haut niveau, la rapidité de la simulation est importante à conserver. La formule obtenue à cette étape formera $G_{ik}(\ldots)$ dans l'équation (3.1). L'équation représentant la relation entre les ASP et les SSP est détaillée à la Section 3.3.

3.3 Élaboration de l'Équation Structurel

À la troisième étape, les résultats des mesures de la Section 3.2 sont utilisés pour construire l'équation $G_{ik}(\ldots)$. Les valeurs obtenues sont analysées pour en extraire la tendance et ainsi établir une équation qui permettra d'estimer la consommation de puissance. Cette équation établit l'importance de l'activité de l'IP selon les caractéristiques structurelles de l'IP. $G_{ik}(\ldots)$ dans l'équation (3.1) se construit d'une façon systématique. Puisque le modèle est utilisé dans un simulateur haut-niveau, il est possible de définir quel paramètre sera utilisé comme variable ou coefficient. La structure de l'IP est définie avant la simulation, car elle modifie la quantité d'activité ou la charge associée à une activité. Ceci permet de définir les coefficients d'une équation par rapport au paramètre de la structure. Ainsi, les paramètres ASP sont ensuite utilisés comme les variables de l'équation qui utilise les coefficients pour produire la consommation. Puisque les SSP ont un effet sur l'activité interne causé par les ASP, il faut dans ce cas calculer les coefficients de l'équation pour refléter l'impact de la structure sur le modèle (comme effet sur l'activité interne et la charge). Par exemple, le modèle structurel de

l'IP peut être exprimé par une équation linéaire comme suit :

$$G_{ik}(ASP, SSP) = \sum_{n=\text{premier ASP}}^{\text{dernier ASP}} \big(H_n(SSP * ASP) + O_n(SSP)\big) \tag{3.2}$$

où $H_n(\ldots)$ est la valeur de la consommation pour chaque activité d'ASP. Ce coefficient est calculé avec les SSP qui sont affectés par l'activité de l'ASP. Le coefficient $O_n(\ldots)$ est la valeur de la consommation d'énergie qui se passe même s'il n'y a pas d'activité sur l'ASP.

Les coefficients des équations H_n et O_n sont composés de la relation des paramètres définissant la structure (SSP) qui sont affectés par l'activité de l'ASP. La structure générale de chacun des coefficients est centré autour de la valeur de base (p. ex. une mémoire simple de taille minimale) qui est affectée par des facteurs qui changent l'impact de l'activité. Ainsi, les coefficients $H_n(\ldots)$ et $O_n(\ldots)$ prennent la forme très simple suivante :

$$H_n \text{ ou } O_n = F(\ldots)V_b \tag{3.3}$$

V_b représente la valeur de la consommation de base. $F(\ldots)$ représente l'équation qui permet de modélisé la consommation pour différent paramètre SSP. La valeur V_b est par la suite ditribuée sur l'équation $F(\ldots)$ pour former H_n et O_n.

L'équation (3.2) représente une part importante du modèle, car elle représente le comportement de l'IP pour différente configuration de SSP et sous différentes activités d'ASP.

Il se peut qu'une représentation linéaire du comportement de l'équation (3.2) ne permette pas d'exprimer assez précisément la consommation de puissance de l'IP. Dans ce cas, Il est possible d'exprimer l'équation par un modèle polynomial (ou dans de plus rares cas exponentiel, puissance ou logarithmique). Des termes peuvent être omis si leur élimination du modèle causent une erreur négligeable. Un comportement qui est aléatoire crée des termes qui habituellement n'ont pas d'impact majeur sur le résultat. De plus, les termes ne pourront représenter le comportement aléatoire car ceux-ci changeront de comportement de fois en fois. Il est préférable d'éliminer ces termes qui ne font qu'alourdir la simulation et qui peuvent ajouter une erreur supplémentaire par rapport à une équation plus simple, mais plus proche du comportement général de l'IP.

L'équation structurelle se forme en considérant les valeurs les plus simples pour les SSP ; ainsi les coefficients $H_n(\ldots)$ et $O_n(\ldots)$ sont considérés comme des constantes. Ensuite, des mesures sont effectuées sur différentes configurations de SSP, les coefficients $H_n(\ldots)$ et $O_n(\ldots)$ sont modifiés en sous-équation pour pouvoir refléter les changements qu'apportent les différentes valeurs de SSP. Une fois que l'équation (3.2) est complétée et qu'elle correspond aux mesures effectuées avec une erreur acceptable, il est possible de commencer l'évaluation des

paramètres d'implémentation TP. Si l'erreur est trop grande, ou si l'équation ne modélise pas le comportement, il faut réévaluer les mesures effectuées et les paramètres sélectionnés.

Les mesures pour la caractérisation sont effectuées par des simulateurs de langage de description matériel comme Modelsim [18, 19] et des estimateurs de puissance RTL comme Xpower de Xilinx [58]. Dans cette étape, les effets du placement et routage ne sont pas pris en compte. Ceux-ci seront rajoutés à la Section 3.4. La caractérisation de l'IP s'effectue avec une simulation structurelle (une simulation comportementale qui contient la structure logique de l'IP). De cette façon, il est possible d'obtenir un profil de consommation de l'IP par rapport à sa structure, mais en ne tenant pas compte de l'implémentation (Placement et Routage) et de la technologie ciblée. À ce niveau de simulation, la fréquence n'a pas d'effet puisque les délais dûs à la logique ou au routage sont considérés comme nuls. La fréquence n'a que pour effet de définir le temps de la simulation. Néanmoins, pour que le choix de la période d'horloge n'affecte pas l'équation structurelle, une mesure en Joules est privilégiée. Une mesure en puissance serait affectée par le temps d'exécution et intégrerait la période d'horloge dans le modèle. De plus, en ayant une notion de temps dans l'équation comportementale sans avoir d'observation sur le routage ne ferait que complexifier l'équation totale et dupliquer l'information à la Section 3.4. La valeur d'énergie obtenue à cette étape n'a évidemment pas de valeur réelle, car elle ne modélise pas toutes les caractéristiques du circuit. Les valeurs représentent l'importance de chacun des paramètres dans la consommation de l'IP par rapport à l'activité et aux caractéristiques structurelles. L'équation $F_k(\ldots)$ multipliera l'équation structurelle pour incorporer les effets des paramètres TP qui sont présenté à la Section 3.4.

3.4 Impact des Paramètres Technologiques et Fréquentiels

À la quatrième étape, l'impact de l'implémentation (fréquence, technologie) est quantifié afin de compléter l'équation. Les Sections 3.2 et 3.3 ont permis de définir l'équation qui modélise l'effet de la structure et de l'activité sur la consommation. Par contre, le placement et routage n'a pas été tenu en compte dans ces équations.

Pour évaluer l'impact du placement et routage, il faut utiliser les paramètres TP qui considèrent la fréquence ciblée, la technologie et la structure (pour des questions de routage et de taux d'utilisations du FPGA). Les mesures sont faites avec les mêmes options pour les ASP et SSP pour bien isoler l'effet des paramètres TP. Puisque l'activité est la même, il n'y a que l'effet de l'implémentation (TP) et de la structure (SSP) qui fait varier la consommation par rapport aux mesures faites à la Section 3.2. La structure de l'IP (SSP) affecte le routage du système et cela peut être mesuré avec une simulation temporelle qui considère les délais

associés au placement et routage.

Les mesures se font en ajustant les paramètres TP et SSP à leur valeur de base pour ainsi permettre d'évaluer séquentiellement l'impact de leur variation sur la consommation. Ceci permet donc d'estimer l'impact des SSP et TP indépendamment de l'activité (ASP) et aussi de minimiser le nombre de mesures pour arriver à une équation complète. En éliminant tous les cas de figure d'activités (ASP) possibles, on minimise le nombre de simulations pour chaque valeur de TP et de SSP. Cette séparation permet la possibilité de réutiliser l'équation (3.2) pour une nouvelle technologie qui utilise le même IP. Puisque la structure même de l'IP et son activité ne change pas, l'équation (3.2) est encore valide comme base pour une nouvelle technologie. Il suffit de refaire les mesures de la consommation de puissance avec les SSP et TP pour la nouvelle implémentation.

Les résultats sont réutilisés avec leur configuration d'SSP et TP correspondant pour former $F_k(\ldots)$. Les valeurs sont comparées aux mesures faites dans la Section 3.2 pour extraire l'effet de l'implémentation sur la consommation. Pour rester cohérent avec les mesures faites précédemment, il faut effectuer les comparaisons en Joules. En changeant la fréquence on change la quantité d'activité en une unité de temps, ceci permet d'éliminer l'effet du temps dans les résultats. Pour un IP qui a la même quantité de transition sur ses signaux, une mesure en puissance donne une valeur différente, puisque le temps change. Le changement de fréquence change le placement et routage assez pour que le nombre de transitions ou une activité parasite (micro-impulsion) augmente ou diminue sur un signal de certains IP. Lorsque la frquence seul change la consommation d'un IP elle devient un paramètre de $F_k(\ldots)$ et non plus un facteur influençant le temps de la simulation pour l'équation (3.1).

3.5 Élaboration de l'Équation Finale

À la cinquième et dernière étape, l'équation $F_k(\ldots)$ est extraite à partir des mesures de l'influence des paramètres SSP et TP de l'IP effectué dans la Section 3.4. $F_k(\ldots)$ multipliera l'équation (3.2) pour obtenir un modèle complet.

Les paramètres technologiques sont utilisés comme un facteur correctif à l'équation structurelle (3.2). Les paramètres technologiques ont un effet qui modifie énormément la consommation à cause de l'ajout d'une information (le placement et routage) qui affecte énormément le modèle. L'équation (3.2) ne considère pas la charge réelle de l'IP qui est causé par le placement et routage. De plus, en ayant de l'information sur les délais des signaux et la charge réelle appliquée sur ceux-ci, il est possible de connaître les transitions aléatoires (micro-impulsion) qui sont causées par l'implémentation. Cette information ne fait que modifier le modèle pour correspondre aux conditions d'implémentation auxquelles l'IP est soumis.

L'équation représentant l'effet de la technologie est généralement relativement simple car elle ne tient compte que de la fréquence choisie (si elle a un effet sur la consommation) et de la technologie d'implémentation. Par contre, si l'IP est complexe, contient beaucoup de routages, utilise des ressources limitées ou utilise beaucoup de ressources dépendant des options choisies, il faut alors ajouter les paramètres SSP pour pouvoir tenir compte de ces facteurs. À cause de la nature des paramètres analysés, il est fort probable que les équations ne soient pas de simples équations linéaires, mais souvent des équations de puissance pour la fréquence ou logarithmique dans le cas d'IP avec un grand nombre de routages.

Puisque les paramètres TP et SSP sont relativement orthogonaux (la fréquence est un effet qui se rajoute à la grosseur de la mémoire), leurs effets seront majoritairement multiplicatifs. L'équation se présentera généralement de la façon suivante :

$$F_k(SSP, TP) = \left(\frac{A}{(Clk_{periode})^n} + B \right) (F_{implementation}) S(SSP) I(TP) \tag{3.4}$$

La diminution de la période de l'horloge permet d'augmenter la consommation de base pour une implémentation sur une technologie ciblée. Les sous-équations $S(\ldots)$ et $I(\ldots)$ permettent de modéliser l'effet des paramètres SSP et TP sur la consommation de puissance. Les paramètres A et B permettent d'ajuster l'effet de la périodes de l'horloge sur l'équation.

L'ensemble des deux sous-équations de l'équation (3.1) forme le modèle de l'IP. Pour expliquer comment la méthode s'applique à différents circuits, le prochain chapitre s'attarde aux différents IP analysés et détaille les particularités de leur modélisation. Ceci permet de présenter des exemples et de montrer comment obtenir un modèle simple tout à la fois précis pour des IP très différents, tant en structure qu'en fonctionnalité.

CHAPITRE 4

APPLICATION DE LA MÉTHODOLOGIE

Dans ce chapitre, la méthodologie est détaillée pour différent modules d'un système de base. Bien que la méthode ait été détaillée dans son ensemble, elle ne peut couvrir tous les cas particuliers qui seront rencontrés avec les différends IP qui existent. Un processeur et une minuterie possèdent un fonctionnement très différent l'un de l'autre, le premier est ponctuel (instruction à chaque cycle) et le second est continu (décompte) avec un état transitoire cyclique (réinitialisation du décompte). Cette grande disparité permet d'évaluer la méthode comme outil de modélisation de consommation de puissance et d'observer où et quand certaines particularités d'un IP peuvent être utilisées pour simplifier l'équation.

4.1 Survol du Système

Comme illustré à la Figure 4.1, le système embarqué typique comprend un processeur, un bus et ses périphériques. Ces derniers sont utilisés pour vérifier la validité du système sur le FPGA et ainsi s'assurer de la validité des simulations et des valeurs obtenues.

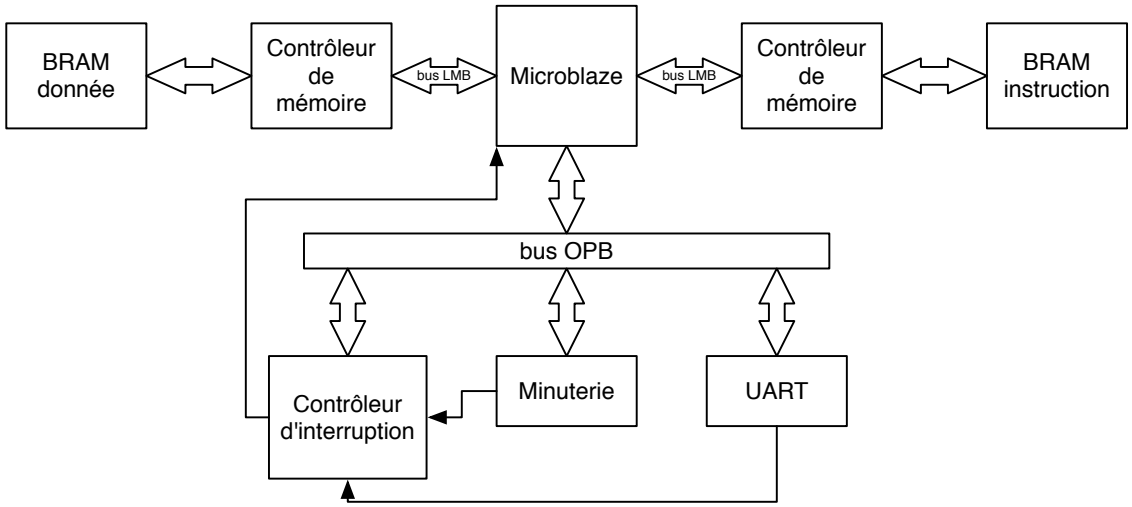


Figure 4.1 Système de base

Le système est développé avec EDK 10.1 de Xilinx [54, 56] et la cible FPGA est le Virtex-II XC-2000[52, 53]. L'outil Modelsim 6.3c est utilisé pour effectuer les simulations RTL au niveau structurel et post-placement et routage. Les mesures de consommation de

puissance RTL de références sont effectuées avec l'outil XPower de Xilinx [58].

Les prochaines sections présenteront le travail effectué sur chacun des IPs du système de base (excepté l'UART) avec les observations particulières lors de leur analyse.

4.2 Script de la méthodologie

Pour faciliter la récolte des données et le contrôle, des outils de mesures et des scripts ont été développés avec le langage Perl [44]. Deux scripts principaux ont été développés. Un premier script s'occupe de contenir la liste des paramètres à analyser et la plage de valeur pour chacun. Ce script contrôle la génération du système EDK en générant différentes architectures et modifie le code C pour que le processeur génère les bons vecteurs de stimulation. Après avoir synthétisé le système, le modèle structurel ou temporel du circuit est utilisé par Modelsim pour exécuter la simulation. Le script récolte les données dans un fichier traité par un analyseur qui groupe la consommation de puissance de chacun des signaux de chaque module. Les valeurs obtenues seront colligées et formatées dans des fichiers texte pour fins d'analyse.

Une fois l'analyse effectuée, le script est utilisé sur un autre système, pour générer une simulation et la consommation de puissance. Les paramètres sont choisis au hasard pour mesurer la qualité du modèle de consommation.

La validation du modèle se fait avec le second script qui prend le fichier d'activité de la simulation (Variable Change Dump (VCD)) et en extrait les transactions. La raison de ce script est de pouvoir obtenir l'information de l'activité dans un format équivalent au modèle transactionnel (TLM) de l'outil Space CoDesign. Au fur et à mesure que l'information est extraite, le modèle calcule la consommation d'énergie. À la fin, la consommation de puissance est estimée et celle-ci est comparé par rapport à la consommation obtenue avec par XPower pour l'IP. Une vue d'ensemble du flot est détaillée dans la Figure 4.2.

4.3 Minuterie

La minuterie permet au processeur de réaliser une horloge de surveillance ou de chronométrer le passage du temps d'un événement [51]. La minuterie est composée d'un compteur, de registres de contrôles et d'un générateur d'interruption. Les registres de contrôles sont utilisés pour configurer le module en compteur ou en chronomètre, activer ou désactiver le circuit de comptage, inscrire la valeur d'initialisation, la direction du décompte et la gestion des interruptions.

L'activité de la minuterie se divise en deux parties principales : la première est le décompte de la minuterie et la seconde est l'accès à la minuterie par le processeur. Le modèle doit tenir

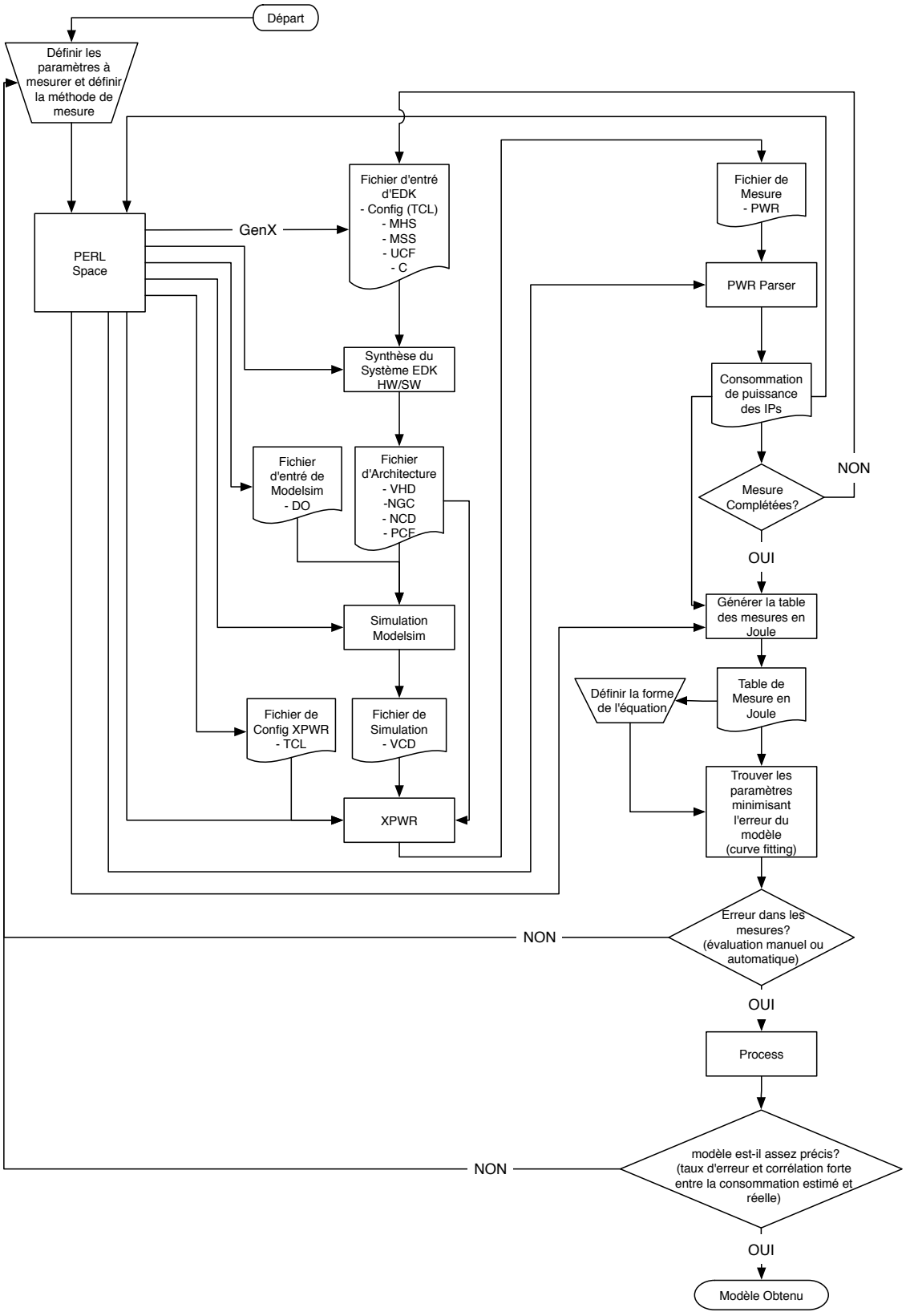


Figure 4.2 Survol du flot d'analyse

compte de ces deux types d'activités.

4.3.1 Sélection des Paramètres

La minuterie effectue durant son fonctionnement le décompte de son compteur et lorsqu'il arrive à la valeur de fin (s'il est en mode circuit à retard et non en mode chronomètre) il active un signal d'interruption. Si la minuterie est configuré avec une valeur de réinitialisation, il recommence son décompte immédiatement. Par la suite, le processeur vient réinitialiser le signal d'interruption.

Si la minuterie est en mode horloge de surveillance, elle sauvegarde le temps d'un événement et envoie une interruption au processeur pour qu'il vienne lire la valeur de temps du compteur. Dans les deux cas, le processeur peut interrompre la minuterie et arrêter le décompte, ainsi il n'y a plus d'activité dans la minuterie puisque la valeur de décompte ne change plus.

La minuterie ne comporte pas de paramètre SSP à proprement parlé. Il n'y a pas vraiment de paramètre influençant son architecture. Par contre, le délai de décompte peut être utilisé pour influencer l'importance d'une de ses activités (c.-à-d. le temps de décompte). Cette valeur peut être considérée comme un paramètre SSP reprogrammable, mais qui ne change pas durant le fonctionnement du compteur.

Voici la sélection des paramètres qui a été réalisé.

ASP : nombre d'interruption ou accès

SSP : nombre de cycle de décompte

TP : technologie d'implémentation, période d'horloge d'opération

On peut s'attendre en observant les paramètres que le modèle peut être représenté par une équation simple.

4.3.2 Mesure des Paramètres ASP et SSP

Pour mesurer l'influence des paramètres ASP et SSP sur la consommation de puissance, le banc de mesure utilise le processeur pour configurer et accéder la minuterie (voir la Figure 4.3). Un script est utilisé pour modifier le code C du processeur afin de changer le décompte de la minuterie. Pour effectuer les mesures correctement, le fichier VCD commencera quand le compteur sera activé et récoltera des données pendant le décompte et l'accès au compteur. Les mesures se feront durant un temps variable pour contenir un nombre différent de décomptes et refaites pour plusieurs périodes de décomptes.

Dans les deux modes, il y a toujours un décompte, une interruption et un accès au travers du bus OPB. Les deux dernières actions requierent un temps généralement beaucoup plus

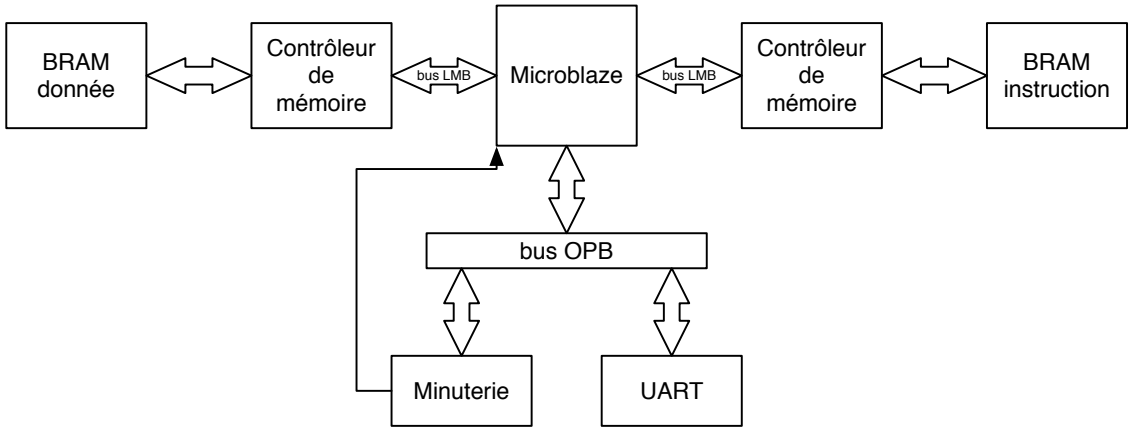


Figure 4.3 Système de mesure de la minuterie

court que le décompte. De plus, le taux d'activités de ces actions n'est pas très élevé dans la minuterie. Donc, si l'erreur dans l'estimation de la consommation de puissance n'est pas trop élevée, il est possible de les regrouper comme une seule action.

4.3.3 Équation Structurelle

Pour la minuterie, l'équation est formatée de façon à représenter le coté configurable du compteur (longueur du décompte) et sa partie fixe (l'interruption et l'accès par le bus). Si la longueur du décompte change durant la simulation, il faudra pouvoir en tenir compte. Voici l'équation structurelle obtenue pour cet IP.

$$G(ASP, SSP) = \sum_{i=1}^{\substack{nbr\,cycle \\ different}} n(AD_i + B) \qquad (4.1)$$

avec

$$n \in [1, \infty[$$

où A représente la valeur de la consommation durant le décompte, B contient la consommation de la transaction effectuée par un accès sur le bus. D_i donne le nombre de périodes d'horloge que dure le décompte du temps entre deux accès au bus. La valeur n indique le nombre de période D_i qui a été effectué durant la simulation.

Donc, pour le premier accès avant que le décompte soit activé D_i est égal à 0. Puisqu'il n'y a pas eu de décompte, on doit tenir compte seulement de la consommation d'accès du bus B. Inversement, si l'on a un décompte continu avec des accès ponctuels, on met la période de décompte entre deux accès dans D_i. Ensuite, on multiplie la valeur obtenue par n (le nombre

de fois que cette période est répétée, sinon on met le nouveau D_i).

4.3.4 Mesure des Paramètres TP et Équations Finales

Pour éviter que les mesures prennent trop de temps, il faut minimiser le nombre de mesures pour chacune des périodes d'horloge et longueur de cycle. Donc, un nombre limité de période de décompte est choisi (centaines, milliers et million). Ceci permet d'avoir une vue d'ensemble du comportement, mais sans faire trop de longues simulations.

Le comportement de l'équation structurel est relativement simple et les résultats obtenus durant les mesures temporelles donnent un comportement similaire, mais avec des constantes différentes. L'équation résultante est très simple :

$$F_k(SSP, TP) = C_k \tag{4.2}$$
$$\text{avec}$$
$$k = a \text{ ou } b$$

où C_k représente la constante multipliant le paramètre de l'équation (4.1) correspondant à l'indice. Par exemple, l'équation $F_a(SSP, TP)$ multipliera la constante A de l'équation (4.1). La consommation donne l'équation finale suivante :

$$M(ASP, SSP, TP) = F_k(SSP, TP)G(ASP, SSP) \tag{4.3a}$$
$$M(ASP, SSP, TP) = F_i(SSP, TP)\left(\sum_{i=1}^{\substack{nbr\,cycle\\different}} n(AD_i + B)\right) \tag{4.3b}$$

ce qui donne

$$M(ASP, SSP, TP) = \sum_{i=1}^{\substack{nbr\,cycle\\different}} n(F_a(SSP, TP)AD_i + F_b(SSP, TP)B) \tag{4.3c}$$

4.4 Contrôleur d'Interruptions du Processeur

La minuterie présentée à la Section 4.3 a son signal d'interruption connecté au processeur au travers d'un PIC si plus d'un signal d'interruption doit être envoyé au processeur. Puisque le processeur n'a qu'un seul port d'interruption disponible et qu'il arrive souvent que le système ait besoin de deux interruptions ou plus (Section 4.1), il faut un module qui puisse avertir le processeur quand il y a une interruption. Le PIC reçoit l'interruption d'un

autre module qui veut transmettre son interruption au processeur. Celui-ci sauvegarde l'interruption et s'il n'y a pas d'autre interruption en attente, il le transmet. S'il y a déjà une interruption en attente, le PIC sauvegardera l'interruption et en retransmettra une nouvelle au processeur quand celle en cours sera traitée.

4.4.1 Sélection des Paramètres

Le PIC est un module qui a la fonction de centraliser les interruptions de plusieurs IP et de les retransmettre au processeur. Pour cela, il utilise de la logique séquentielle et un encodeur de priorité de base. Le numéro du port d'interruption détermine sa priorité (le port LSB à la priorité la plus élevé). Ceci implique que le PIC est un IP très régulier, car chaque interruption en entrée implique la même logique ajoutée à l'interne. Il est possible de définir l'activité des interruptions assez facilement car ils sont interchangeables.

Cette grande symétrie dans le PIC permet de supposer que la ligne d'où provient l'interruption est plus ou moins importante. Ceci nous donne le groupement de paramètres suivants :

ASP : nombre d'interruptions reçus

SSP : nombre de ligne d'interruptions

TP : technologie d'implémentation, période d'horloge d'opération

4.4.2 Mesure des Paramètres ASP et SSP

Pour mesurer efficacement la consommation d'énergie du PIC, il faut être capable de contrôler l'activité à l'entrée. Pour cela, il n'est pas possible de brancher plusieurs IP au PIC et tenter d'estimer la consommation d'énergie. Il faut être capable de contrôler la quantité d'interruption qui est envoyée et pouvoir connecter/déconnecter des signaux d'interruption. Pour cela, un module a été créé pour permettre le contrôle de la génération d'interruption. Le module génère des interruptions après un interval prédéfini par une constante qui est contrôlée grâce au script Perl. Ceci permet de contrôler le nombre d'interruptions qui sera envoyé et permet de limiter la longueur des simulations en augmentant la densité d'interruption. De plus, le script s'occupe de connecter les signaux d'interruption du générateur au PIC pour changer le paramètre SSP.

Pour ne pas avoir des interruptions non voulues, la minuterie et l'UART ont été retirés du système lors des mesures. Le processeur est utilisé pour configurer le PIC et interagir avec celui-ci quand il activera son signal d'interruption (voir la Figure 4.4). Puisque nous voulons mesurer la consommation de puissance du PIC, le générateur d'interruption n'a pas de connexion au bus. Normalement, après avoir consulté le PIC pour déterminer l'IP

qui a envoyé l'interruption, le processeur communique avec l'IP indiqué. Dans notre cas, le processeur va éteindre l'interruption et retourner en attente d'une nouvelle interruption.

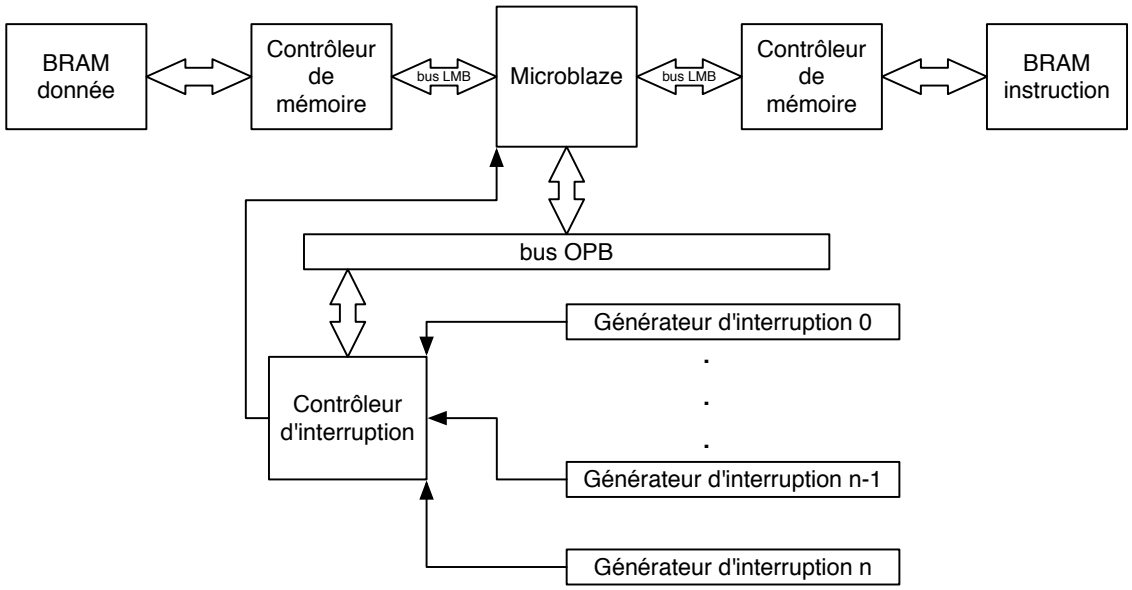


Figure 4.4 Système de mesure du PIC

Au cours des mesures, il a été possible de constater que la consommation des interruptions est similaire, peu importe d'où elle provient, puisqu'il n'y a pas de circuit de discrimination et que la priorisation des interruptions se fait par la position dans le vecteur d'interruption (définit par le numéro du port d'interruption d'entrée du PIC). Le traitement d'une interruption se fait de façon identique, la consommation obtenue est donc la même. L'impact le plus grand est le nombre de connexions, car EDK définit la valeur des paramètres génériques (en VHDL) de l'IP avec le nombre de ports d'interruption demandé. S'il y a 5 interruptions connectées au PIC, EDK configurera l'IP pour n'implémenter que de la logique et des registres pour les ports présents. Donc, seule la logique de contrôle reste présente, peu importe le nombre d'entrées. L'équation se concentrera sur le nombre de signaux d'interruption connectés et au nombre d'interruptions détectées.

4.4.3 Équation Structurelle

Comme pour la minuterie (voir Section 4.3), le modèle du PIC représente l'interruption et l'accès au module par le bus OPB de façon simplifiée. Puisqu'une interruption est suivie d'un accès au travers du bus OPB, l'équation regroupera l'activité pour focuser sur le nombre de connexions et le nombre d'interruptions. Voici le format de l'équation :

$$\begin{aligned} G(ASP, SSP) &= I(AC + B) \text{ avec} \\ I &\in [1, \infty[\end{aligned} \tag{4.4}$$

où A représente la consommation de chaque connexion durant l'utilisation du PIC (interruption et communication au travers du bus). B représente la consommation constante du PIC. C correspond au nombre de connexion d'interruption au PIC et I le nombre d'interruptions qui sont traitées.

Le PIC a comme particularité qu'il n'y a aucune activité dans l'IP tant qu'une interruption n'est pas détectée. Ainsi, la période de temps entre deux interruptions n'est pas importante. Cette particularitée simplifie grandement l'équation du modèle en permettant de structurer une équation indépendante du temps mort.

4.4.4 Mesure des Paramètres TP et Équations Finales

La mesure de l'impact des paramètres SSP et TP se limite à refaire les mesures de la Section 4.4.2. Pour ne pas prendre trop de temps de simulation, les mesures ne seront pas faites pour chaque nombre de connexion d'interruption. D'après l'équation (4.4) trois points seront suffisants : une mesure avec deux connexions, une avec 32 connexions et la dernière avec 16. Ces trois mesures permettent de s'assurer que la consommation d'énergie reste toujours linéaire. Il faut faire des mesures avec quelques fréquences de fonctionnement pour s'assurer de représenter la consommation d'énergie correctement.

Les mesures montrent que la fréquence a un impact sur la consommation d'énergie. Le PIC contient beaucoup de connexions à d'autres modules esclaves et au processeur. De plus, il contient plusieurs registres qui contiennent l'état interne des interruptions et l'état de l'IP. Ceci cause le PIC à être sensible aux contraintes de PAR, puisqu'une des méthodes efficaces pour répondre aux contraintes de fréquence est de minimiser la longueur des connexions. Voici l'équation obtenue :

$$F_k(SSP, TP) = C_k P + D_k \tag{4.5}$$

avec

k=A ou B

(4.6)

où B représente le facteur correctif de base pour avoir nos mesures de consommation en temporel, A permet de corriger les valeurs de l'équation (4.4) avec une période d'horloge en ns avec laquelle on fournit le PIC, P est la période de l'horloge en ns. Comme avec l'équation (4.3), la fonction utilise l'indice k pour multiplier le paramètre de l'équation (4.4) voulu. Voici l'équation finale.

$$M(ASP, SSP, TP) = F_k(SSP, TP)G(ASP, SSP) \tag{4.7a}$$

$$M(ASP, SSP, TP) = F_k(SSP, TP)\left(I(AC + B)\right) \tag{4.7b}$$

ce qui donne

$$M(ASP, SSP, TP) = I(F_A(SSP, TP)AC + F_B(SSP, TP)B) \tag{4.7c}$$

4.5 Mémoire

La mémoire utilisée est une «Block Random Access Memory» (BRAM) intégrée dans le FPGA qui permet de créer une mémoire dans l'ordre des KB en ayant un minimum d'impact sur la performance et le routage.

La BRAM peut-être utilisée dans un circuit comme mémoire tout usage (FIFO ou mémoire de circuit), dans notre cas elle est utilisée avec un processeur (voir Section 4.6). Elle sert de mémoire d'instruction et de donnée pour le programme. Pour que le processeur (Microblaze) puisse accéder la mémoire il faut aussi utiliser un «Local Memory Bus» (LMB) qui permet de raccorder plusieurs blocs mémoire. Un contrôleur de mémoire n'a pas la capacité de gérer une plage mémoire supérieure à 64 KB et il faut que la plage d'adressage soit une puissance de 2 (2^n). Si le besoin en mémoire est suppérieur à 64 KB, il faut avoir deux ou plusieurs contrôleurs de mémoires avec une BRAM pour chacun. Notre analyse de la mémoire modélisera la BRAM, les contrôleurs et le bus LMB.

4.5.1 Sélection des Paramètres

Cet IP ne performe que très peu d'action différentes. Elles se résument à la lecture, à l'écriture de donnée et, si aucune action n'est performé, à rester inactif. Par contre, plusieurs paramètres affectent sa structure et son implémentation. Ceci est dû au fait que la mémoire est composée de registres et de multiplexeurs. Donc, si l'on change la grosseur de chacun des blocs mémoires ceci affectera le placement et routage de ce bloc. En ajoutant un bloc mémoire supplémentaire on ajoute un contrôleur ce qui complexifie le bus LMB, car il faut être capable de connecter toutes les composantes. Le coeur de ce modèle se situera dans la capacité à modéliser les différentes configurations des paramètres SSP pour que les paramètres ASP donnent la bonne consommation.

Les paramètres sont regroupés selon cette configuration :

ASP : valeur de la donnée, valeur de l'adresse, le type d'accès (lecture ou écriture) et cycle inactif

SSP : grosseur du bloc mémoire, le nombre de bloc de mémoire, largeur du bus de donnée, largeur du bus d'adresse et accès mémoire avec un ou deux ports

TP : technologie d'implémentation, fréquence d'opération

Ces paramètres correspondent aux caractéristiques d'un grand nombre de mémoires intégrées. Rapidement, on peut voir que les paramètres structurels auront un grand impact sur le placement et routage, donc un effort accru devra être mis durant les mesures pour cerner leur effet.

4.5.2 Mesure des Paramètres ASP et SSP

Pour analyser l'impact des paramètres ASP et SSP sur la consommation de puissance, le processeur est utilisé pour stimuler la mémoire. En utilisant le processeur, on s'assure que l'activité mesurée correspond au fonctionnement normal du système. Un script modifie les paramètres structurels de la mémoire dans le projet EDK. De plus, pour obtenir l'activité voulue, le même script modifie l'application exécutée par le processeur, ceci nous permet de calculer le temps de simulation pour obtenir un nombre entier de transactions.

Les valeurs de la donnée et de l'adresse sont mesurées au niveau du bus LMB puisque le modèle dans l'outil Space Codesign[33] ne permet pas de faire la simulation de composantes intermédiaires comme le contrôleur. Le modèle devra tenir compte des contraintes des modules (spécifié à la Section 4.4.4) utilisés pour pouvoir obtenir un modèle représentatif de l'architecture de la mémoire implémentée.

Pour une lecture ou une écriture, il faut analyser les données transmises entre la mémoire et le processeur. Lors d'une lecture, le processeur envoie l'adresse, la donnée ne change pas

ou est remise à zéro. Au retour, la valeur voulue est retournée au processeur avec les adresses mises à zéro s'il n'y a pas d'autre lecture. Pour l'écriture, le processeur envoie la donnée et l'adresse durant le même cycle.

Pour simplifier l'analyse, la lecture et l'écriture sont analysées indépendamment. De plus, pour faciliter le calcul de l'impact de la consommation de la transition des signaux d'adresse et de donnée, différents accès seront alternés. Par exemple, pour déterminer la consommation d'un bit d'adresse, la même donnée sera écrite à l'adresse 0x0 et 0x1. Pour un bit de donnée, ce sera la donnée 0x0 et 0x1 qui sera écrite à la même adresse. Puisque l'envoi et la réception de données de la mémoire se font sur différents signaux, les accès peuvent être intercalés pour ne pas avoir de cycle mort (période de temps durant laquelle il n'y a pas d'accès à la mémoire). Pour évaluer la consommation des signaux de contrôle, on alterne une lecture et une écriture de la même valeur à la même adresse (ici la donnée est égale à zéro). Ces mesures sont aussi refaites avec des temps morts pour voir l'impact de l'arrêt des accès.

Par contre, les mesures ont montré que la lecture et l'écriture sont identiques sauf pour les signaux de contrôle. Les deux envoient la valeur contenue dans le registre spécifié dans la commande assembleur (le registre d'envoi pour l'écriture et de réception pour la lecture). Ensuite, la mémoire retourne la valeur lue dans le registre. Les signaux ne font que bloquer l'écriture dans la mémoire ou dans le registre du processeur. Ceci cause certains problèmes durant la validation de l'équation, car durant la simulation du modèle transactionnel d'une lecture, il n'y a normalement pas de données envoyées à la mémoire et inversement pour l'écriture. Cette activité devra être considérée par le modèle au niveau TLM pour causer le moins d'erreurs possibles.

Une limitation se présente lors de l'analyse des cycles mort (cycle sans accès). Dans le modèle VHDL de l'IP, il est possible de voir que les ports du Microblaze ne sont pas forcés à zéro lors d'un cycle sans accès au bus. Selon l'instruction exécutée par le processeur, il est possible de voir des valeurs provenant d'un calcul ou du bus d'instruction (si on analyse la mémoire de donnée). La raison est que le port du bus et de la mémoire de donnée sont connectés au bus interne de donnée du processeur sans logique de séparation. L'outil Space Codesign n'effectue pas la simulation de ces cycles pour la mémoire, car elle ne participe pas au fonctionnement du système. Le saut des cycles morts permet d'accélérer la simulation TLM et ainsi de faire des simulations complexes de système sur puce en un temps raisonnable pour l'exploration. Le modèle n'a pas d'information pour ces cycles, donc un modèle d'approximation d'activité est nécessaire pour permettre au modèle de diverger le moins possible de la réalité.

Effectuer une moyenne des valeurs de lecture et d'écriture à la mémoire fournit une approximation de l'activité inconnue détaillée dans les paragraphes ci-dessus. Puisque les valeurs lues et écrites ont généralement un lien avec les valeurs calculées, il est sensé de considérer

qu'une moyenne peut être une bonne approximation de l'activité. Si le nombre de lecture et d'écriture ne sont pas assez nombreux pour effectuer une bonne approximation, le modèle aura malheureusement tendance à diverger de la réalité, mais ceci peut être très rapidement compensé en ayant plusieurs activités d'accès à la mémoire.

Comme il a été mentionné dans la Section 3.2, les mesures obtenues peuvent parfois contenir des artefacts ou donner une tendance très complexe à représenter avec une équation. Dans ce cas, il faut voir s'il est possible d'exprimer la tendance avec une équation plus simple en minimisant la déviation. Par exemple, lors de la mesure d'énergie du bus LMB sous différentes configurations de la mémoire, il est possible d'observer que le nombre de bits de donnée ou d'adresse ne fait pas varier la consommation tout à fait linéairement. À la Figure 4.5, il est possible de voir la courbe obtenue par rapport à deux droites représentant une croissance linéaire. Bien que les courbes s'approchent des deux droites de référence, chacune possède une déviation par rapport à l'une ou l'autre des droites de référence. Par rapport aux droites, les points qui divergent ont en moyenne un écart inférieur à 20%. Le total des erreurs se centre autour des droites, autant supérieures qu'inférieures, sans former une tendance générale interprétable facilement en une équation. Donc, l'approximation linéaire offre une équation qui permet d'avoir des estimations proches de la réalité, compte tenu des conditions.

4.5.3 Équation Structurelle

L'équation utilise une approximation du comportement de la consommation dans les situations ou les courbes de consommation des IPs divergent légèrement d'une fonction simple. Il est inutile de fournir une équation complexe si la complexité ajoutée n'augmente pas de façon significative la précision du modèle ou que, pour suivre exactement les données, l'équation demande une complexité élevée.

Pour améliorer la précision du modèle de la mémoire, les différents sous IPs auront chacun leur équation. Ceci permet d'avoir une équation par IP, ce qui simplifie l'équation globale, car l'équation n'a pas à tenir compte de toutes les configurations possibles. Pour chaque IP présent dans le système, on a qu'à rajouter une équation. De plus, certains IP (contrôleur et BRAM) sont réutilisés dans d'autres parties du circuit (mémoire partagée sur le bus OPB). D'autre part, Xpower crée un autre IP supplémentaire durant son analyse de la mémoire qui consiste en la connexion entre le contrôleur et la BRAM. Donc, pour tenir compte de toute la consommation, un IP appelé Port sera ajouté pour tenir compte de cette subdivision de la mémoire par l'outil Xpower. Cette division permettra de maximiser la précision de l'équation dans le contexte de la mémoire. Ce qui permet de négliger certains paramètres qui n'ont pas d'effet.

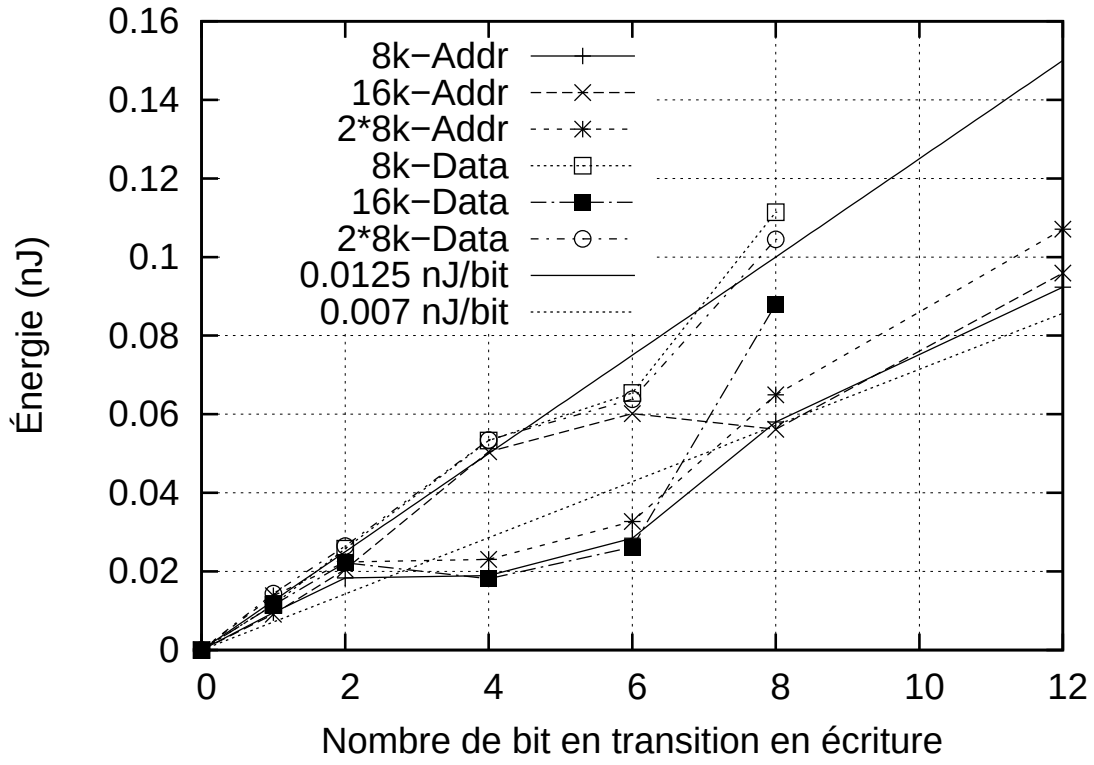


Figure 4.5 Consommation du bus LMB pour différentes configurations de mémoire

De plus, une distinction peut être faite entre la mémoire de donnée et d'instruction. Puisque la mémoire d'instruction n'est jamais écrite, la logique pour l'écriture a été éliminée. En plus d'enlever une variable, elle change la consommation d'énergie de la lecture et du cycle inactif, car le circuit de gestion présente une charge différente qui est présentée aux signaux. Ceci force l'utilisation de deux équations pour représenter la différence dans le comportement de ces deux mémoires.

Voici les équations pour ces deux mémoires :

$$G_I(ASP, SSP) = L_I(ASP, SSP) + \sum_{n=1}^{\text{num of mem}} \big(C_{In}(ASP, SSP) + B_{In}(ASP, SSP)\big) \tag{4.8a}$$

$$G_D(ASP, SSP) = L_D(ASP, SSP) + \sum_{n=1}^{\text{num of mem}} \big(C_{Dn}(ASP, SSP) + P_{Dn}(ASP, SSP) + B_{Dn}(ASP, SSP)\big) \tag{4.8b}$$

L'équation (4.8a) est utilisé pour la mémoire d'instruction (l'indice I), tandis que l'équation (4.8b) est pour la mémoire de donnée (l'indice D). La fonction $L(\ldots)$ représente la consommation du bus LMB, $C_n(\ldots)$ décrit l'activité du contrôleur de mémoire, $P_n(\ldots)$ représente l'IP Port ajouté par la synthèse et $B_n(\ldots)$ est la mémoire BRAM à proprement parler. Les équations résultantes pour la mémoire ont le format suivant :

pour la mémoire d'instruction

$$C_{In}(ASP, SSP) = (K_1 S)^{K_2}\big(K_3 CBit_{addr} R + K_4(Bit_{addr} + Bit_{data})I\big) \quad (4.9a)$$

$$B_{In}(ASP, SSP) = (K_5 S)^{K_6}\big((K_7 Bit_{addr} + K_8 Bit_{data})R + (K_9 Bit_{addr} + K_{10} Bit_{data})I + K_{11}\big) \quad (4.9b)$$

$$L_I(ASP, SSP) = (K_{12} S)^{K_{13}}\big((K_{14} R + K_{15} I)(Bit_{data} + Bit_{addr}) + K_{16}\big) \quad (4.9c)$$

pour la mémoire de donnée

$$C_{Dn}(ASP, SSP) = (K_{17} S)^{K_{18}}\big(K_{19} CBit_{addr} R + K_{20} CBit_{addr} W + K_{21}(Bit_{addr} + Bit_{data})I\big) \quad (4.9d)$$

$$B_{Dn}(ASP, SSP) = (K_{22} S)^{K_{23}}\big((K_{24} Bit_{addr} + K_{25} Bit_{data})R + (K_{26} Bit_{addr} + K_{27} Bit_{data})W + (K_{28} Bit_{addr} + K_{29} Bit_{data})I + K_{30}\big) \quad (4.9e)$$

$$P_{Dn}(ASP, SSP) = (K_{31} S)^{K_{32}}\big((K_{33} R + K_{34} W + K_{35} I)(Bit_{data} + Bit_{addr}) + K_{36}\big) \quad (4.9f)$$

$$L_D(ASP, SSP) = (K_{37} S)^{K_{38}}\big((K_{39} R + K_{40} W + K_{41} I)(Bit_{data} + Bit_{addr}) + K_{42}\big) \quad (4.9g)$$

avec

$$R, W, I, C \in [0, 1]$$

Dans les équations (4.9), les variables R, W et I indiquent une lecture, une écriture ou un cycle mort pour chacun des cycles. Les constants K_1 à K_{42} représentent la quantité d'énergie d'un bit en transition pour les trois activités possibles. Les variables Bit contiennent le nombre de bits qui changent de valeur pour l'adresse et la donnée. Le facteur multiplicatif au début des équations permet de tenir compte de l'impact de la grosseur des mémoires sur la consommation. Finalement, la variable C indique quand il y a eu un accès contigu au précédent (sans cycle mort).

Les cycles mort de tous les modules demande une approximation de l'activité. Cette approximation sera obtenue durant la simulation. Par la suite, il est possible de faire une post-annotation des cycles morts pour leur donner une consommation de puissance. Sinon, il est possible d'utiliser la valeur moyenne calculée au moment du cycle inactif et affecter l'équation immédiatement, ce qui fait varier l'estimation avec l'activité courante dans le temps.

4.5.4 Mesure des Paramètres TP et Équations Finales

Une attention particulière a dû être apportée lors de l'analyse de cet IP. L'agencement du bus avec les différents contrôleurs et les blocs mémoires fait en sorte que le nombre de possibilités est très grand ; et ceci amène un très grand temps de simulation (particulièrement avec une analyse post placement et routage). Pour minimiser le temps de simulation, il faut diviser le problème pour obtenir un résultat en un temps raisonnable.

Bien que la mémoire contient beaucoup d'interconnexions, la fréquence ne semble pas changer beaucoup les résultats de la consommation d'énergie. La grosseur de la mémoire semble être beaucoup plus importante car le placement et routage est affecté par l'espace que prend la BRAM. On n'ajoute pas que de la logique, mais aussi de la mémoire dédié ce qui augmente complexité du routage. Donc, en ajoutant les délais internes du système sur les connexions l'espace requis par la BRAM influence grandement la consommation à cette étape.

Voici le format de l'équation :

$$F_{kl}(SSP, TP) = (CS)^n \tag{4.10}$$
$$\text{où}$$
$$k = L, C, P, B$$
$$l = I, D$$

où S représente la grosseur en KB de la BRAM. C est la constante donnant une valeur au poid de la BRAM sur la consommation. L'exposant n module l'influence de l'accroissement de la grosseur de la BRAM.

Ceci donne comme équation finale :

$$M(ASP, SSP, TP) = F_{kl}(SSP, TP)G_l(ASP, SSP) \tag{4.11a}$$

ce qui donne

$$G_I(ASP, SSP, TP) = F_{LI}(SSP, TP)L_I(ASP, SSP) \tag{4.11b}$$
$$+ \sum_{n=1}^{\text{num of mem}} \left(F_{CI}(SSP, TP)C_{In}(ASP, SSP) + F_{BI}(SSP, TP)B_{In}(ASP, SSP)\right)$$

$$G_D(ASP, SSP, TP) = F_{LD}(SSP, TP)L_D(ASP, SSP) \tag{4.11c}$$
$$+ \sum_{n=1}^{\text{num of mem}} \left(F_{CD}(SSP, TP)C_{Dn}(ASP, SSP) + F_{PD}(SSP, TP)P_{Dn}(ASP, SSP)\right.$$
$$\left.+F_{BD}(SSP, TP)B_{Dn}(ASP, SSP)\right)$$

4.6 Processeur

Un des IP les plus importants est le processeur car il permet l'exécution du logiciel. Dans ce mémoire, le processeur analysé est le Microblaze de Xilinx [55]. Cet IP est dépendant de la mémoire BRAM présentée à la Section 4.5 qui contient le code et les données utiles à son fonctionnement.

Le Microblaze analysé inclu un multiplieur et diviseur 32 bits à point fixe. Par contre, l'unité de calcul à point flottant (FPU) et la mémoire cache n'ont pas été considérées dans cette analyse. Le processeur reçoit des valeurs et des instructions qui dictent son activité et sauvegarde des valeurs dans sa mémoire pour une utilisation future. Ces caractéristique peuvent sembler complexifier l'analyse du processeur, mais en réalité le processeur a une particularitée de laquelle découle son fonctionnent. Un processeur effectue une activité qui est ponctuelle dans le temps et qui est liée seulement par l'instruction précédente dans le pipeline d'instruction.

4.6.1 Sélection des Paramètres

La particularité d'un processeur ou du Microblaze du point de vu structurel consiste en un structure interne composée de mini circuits indépendants. Chacun des circuit de l'Unité Arithmétique et Logique (ALU) fonctionne indépendamment l'un de l'autre (c.-à-d. addition, multiplication, lecture, etc.). Ce qui implique que chacune des instructions (ASP) ont une faible relation entre eux. Si on enlève le multiplieur, l'activité des autres instructions ne change pas. Bien sûr l'activité globale du logiciel change, mais la multiplication ne se fera pas avec une instruction spécialisée. L'addition aura donc la même activité pour la même

addition, avec ou sans l'instruction de multiplication dans l'ALU. Chacune des activités du processeur peut être limitée à un moment ponctuel du début du cycle d'horloge jusqu'à la fin du même cycle.

Cette relative indépendance de l'activité causée par la séparation des sous-circuits permet de découpler l'effet des différents paramètres. Voici la sélection des paramètres :

ASP : instructions, instruction précédente (changement d'instruction), pipeline actif ou inactif et instruction multi-cycle

SSP : instruction matériel spécialisé

TP : fréquence d'opération, technologie et implémentation

Il faut noter que l'adresse et la valeur du registre, lus et écrits à chaque instruction par le processeur, ne sont pas présentes dans les paramètres ASP. Cela se produit car la valeur n'a pas un impact majeur sur la consommation du Microblaze. Une instruction activée a un plus grand impact sur la consommation que la valeur des registres ; ce qui simplifie les équations. Ce phénomène est détaillé dans la Section 4.6.3. Par contre, il faut tenir compte des variations dans le comportement du processeur pour chacune des instructions (p.ex. un saut ou non lors de l'exécution d'un branchement). Ceci implique aussi de tenir compte des instructions comme la division qui utilise plus d'un cycle pour s'exécuter sauf si on divise par zéro. Pour cela, il faut observer l'état du processeur (voir Section 4.6.2). L'analyse s'apparente à la mesure de la consommation des instructions proposées par Tiwari, Malik, Wolfe et Lee [43]. Ils effectuent des mesures de courant pour déterminer la consommation des instructions, du changement d'instructions et tout autres effets inter-instructions (comme l'arrêt temporaire du pipeline). Puisque nous n'avons pas accès au processeur directement il faudra effectuer les mesures avec Xpower ; mais les mesures considéreront des paramètres similaires.

4.6.2 Mesure des Paramètres ASP et SSP

Pour effectuer les mesures automatiquement, un script Perl [44] contrôle les outils EDK, Modelsim et XPower. Le même script permet le changement des instructions et des valeurs dans le fichier source pour contrôler le comportement du processeur. Le script peut générer un code contenant la même instruction exécutée continuellement ou un mélange d'instructions pour évaluer l'effet du changement d'instruction. De plus, si l'instruction prend une ou deux valeurs en entrées le script ajuste le patron de données pour évaluer l'impact de leur valeur sur la consommation et confirmer que leur valeur peut être négligée. Les trois patrons utilisés sont :

i) la même valeur pour chacun des appels d'instruction, ii) valeur qui change en incrémentant ou décrémentant et iii) valeur aléatoire qui couvre le spectre de valeur possible et

voulue.

Différentes combinaisons de ces patrons sont utilisées si l'instruction prend plusieurs valeurs en entrée. Après l'application de ces profils de valeurs sur les entrées des instructions, on peut voir que l'effet de la valeur ou le changement de valeur a très peu d'effet sur la consommation de puissance, tel qu'illustré à la Figure 4.6.

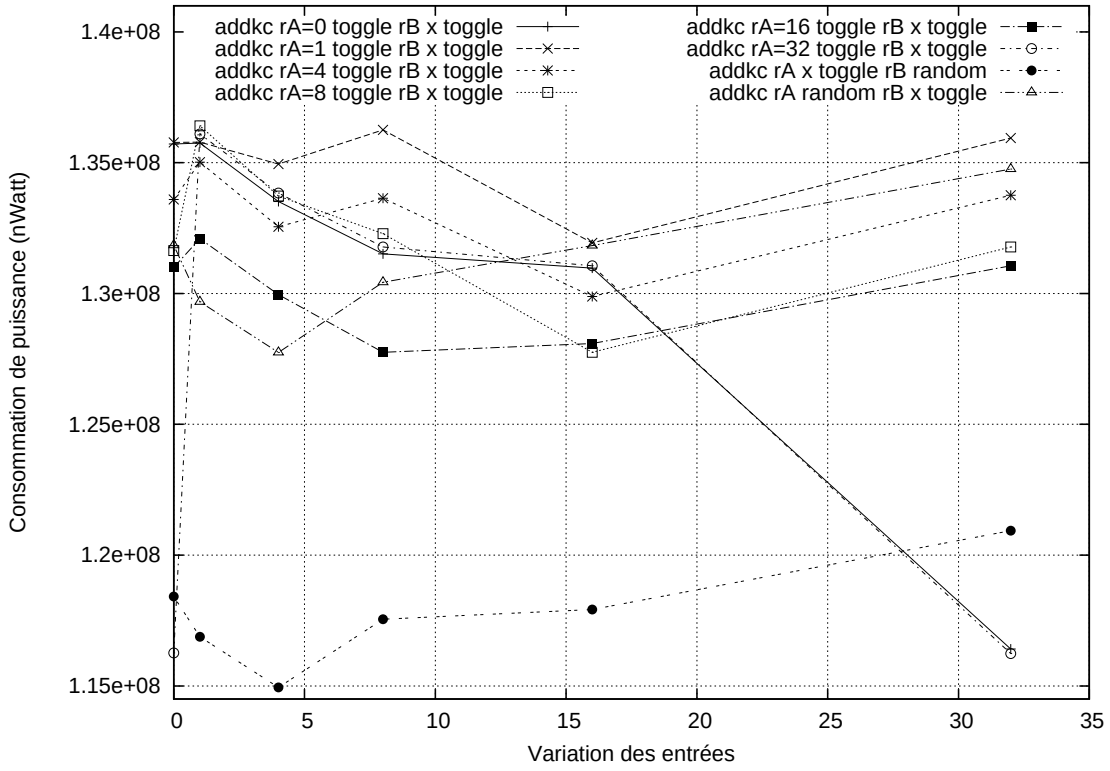


Figure 4.6 Variation de la consommation de puissance pour différents patrons de valeur d'entrée pour l'instruction *addkc*

Dans cette figure, 81.25% (39/48) des valeurs se situe entre 127.5 mW et 136 mW ce qui donne une variation maximale de 7% de la valeur de la consommation. Si on ajoute les 9 (18.75%) valeurs qui se situent à l'extérieur de cette gamme de valeurs, on obtient une variation maximale de valeur de 18.2%. Cette figure montre que la valeur utilisée par l'instruction ne tend pas à faire une grande variation dans la consommation. Elle ne montre que des appels continuels à la même instruction, mais le cas se répète quand le processeur fait appel à deux instructions différentes. Cette caractéristique se reflète aussi sur les autres instructions.

Pour s'assurer que la variation causée par la valeur d'entrée de l'instruction puisse être négligée, une comparaison est fait entre un appel constant de la même instruction et un appel alterné de deux instructions. Deux situations ont été explorées : un appel continu de la même

instruction avec une entrée de valeur aléatoire pour les deux registres et un appel alterné de deux instructions (*addkc* avec un «carry in» sans «carry out» et le *nop*). Dans les deux cas, les valeurs en entrée ont un patron aléatoire. Dans le cas de l'appel continu, la consommation se situe à 140 mW tandis que l'appel d'instruction alterné consomme 200 mW. L'appel de l'instruction *addkc* alterné avec un *nop* cause une augmentation globale de 42.8% ce qui est beaucoup plus important que l'effet de la valeur d'entrée (18.2% pour tous les points et 7% pour 81% des points). Le changement d'instruction ajoute de 10% (pour des instructions similaires, add vs addi) jusqu'à 50% (pour des instructions différentes, mul vs addi). Donc, puisqu'il est possible de négliger la valeur des registres, un patron aléatoire pour tous les registres ou valeurs immédiates est utilisé. Il est important de savoir quand il y aura deux appels consécutifs ou un changement d'instruction et de connaître l'impact de la transition de la consommation surtout pour des instructions qui ont un comportement différent.

Pour des instructions qui ont différents résultats, ou comportement dépendant des valeurs d'entrées, il faut faire l'analyse pour la plage de valeur et recommencer pour chaque comportement (ou plage de valeur donnant le même résultat). Dans le cas d'un branchement, l'analyse pour un saut d'adresse ou non doit être effectué :

4.6.3 Équation Structurelle

Pour le Microblaze, il est facile de cibler l'instruction comme le point central de l'équation. Voici le format de l'équation obtenu pour le Microblaze en utilisant les paramètres ASP et SSP sélectionnés.

$$G(ASP, SSP) = \sum_{k=\text{première instr}}^{\text{dernière instr}} \sum_{s=0}^{\substack{\text{num of} \\ \text{one cycle instr}}} (I_{sk} + D_{sk} d_{sk}) l_{sk} + Pl_{k\,idle} + Ml_{k\,multi} \qquad (4.12)$$

où

$$l_k, l_{idle}, l_{multi}, d_k \in [0, 1]$$

Dans l'équation (4.12) I_{sk} représente l'énergie consommé par l'instruction s (qui ne dure qu'un cycle) au temps k, D_{sk} est la conommation supplémentaire d'énergie si l'instruction k est différente de la précédente $(k-1)$ et d_{sk} indique si l'instruction est la même ou non. Pour indiquer l'instruction active à un cycle donné on se réfère à la variable l_{sk}. Le coefficient P est la quantité d'énergie qui est consommée quand on vide le pipeline après un branchement (le pipeline est inactif à ce moment). Le coefficient M exprime la consommation d'énergie

durant l'exécution d'une division multicycle. Les variables l_{multi} et l_{idle} indiquent si la division en cours d'exécution sur plusieurs cycle ou si le processeur est dans un état inactif (l_{idle}).

Puisque la division est la seule instruction qui est multicycle dans le Microblaze, le coefficient M est utilisé uniquement pour calculer la consommation d'énergie s'il exécute des cycles supplémentaires. Si le processeur vide le pipeline (c.-à-d. après un branchement ou un saut dans l'exécution du code) le coefficient P est utilisé pour calculer l'énergie durant le remplissage du pipeline. Pour savoir ce que le processeur exécute sans utiliser des ports d'accès particuliers à un modèle de processeur, il faut utiliser des signaux et les informations disponibles pour tous les processeurs. Il faut un signal indiquant l'instruction exécuter (pour le Microblaze c'est un signal indiquant les instructions chargées dans le pipeline et un autre indiquant lesquels sont exécutées par le processeur). Par la suite, il faut une façon de détecter les sauts et les instructions multicycles. Une méthode efficace est d'utiliser le Compteur de Programme (PC) pour suivre l'exécution. Si le compteur de programme n'incrémente pas après une division, c'est qu'il calcule la division (une division par «0» n'est pas calculée et ne dure alors un cycle). Si le compteur change de valeur autre que «PC+4» c'est qu'il a effectué un saut d'adresse après un branchement. Il se peut que le compteur de programme n'incrémente pas pour un cycle après le saut car le Microblaze permet de ne pas précharger la prochaine instruction.

4.6.4 Mesure des Paramètres TP et Équations Finales

Pour la mesure des effets des paramètres TP et SSP ensemble, le patron aléatoire avec et sans *nop* est utilisé (car les autres patrons sont inutiles pour la mesure, voir 4.6.2). Pour l'analyse des différentes instructions, il est possible de voir le Microblaze comme un ensemble de circuits qui sera affecté différemment par les paramètres technologiques (ALU, accès mémoires, etc...). Les instructions similaires dans leur fonctionnement auront des paramètres similaires pour la fonction $F(\dots)$ (*add*, *addk*, etc... ; *or*, *and* et *xor*). Ceci permet de minimiser le nombre de paramètres pour la sous-instruction $F_k(\dots)$, malgré le fait que différents groupes d'instructions auront différents paramètres.

Bien qu'il semble que nous utilisons les paramètres ASP dans l'analyse de la fonction $F(\dots)$, il n'en est rien. La particularité est que nous utilisons le circuit matériel qui est relié à l'instruction que nous voulons exécuter. Dans la sélection des paramètres à la Section 4.6.1, l'instruction matérielle spécialisée a été mentionnée comme SSP pour référer au circuit et non à l'instruction lue par le processeur.

Lors de la génération des différentes implémentations, il a été remarqué que pour différentes fréquences la consommation d'énergie reste stable. La Figure 4.7 présente la consommation en Joules de l'instruction *addkc* à différentes fréquences.

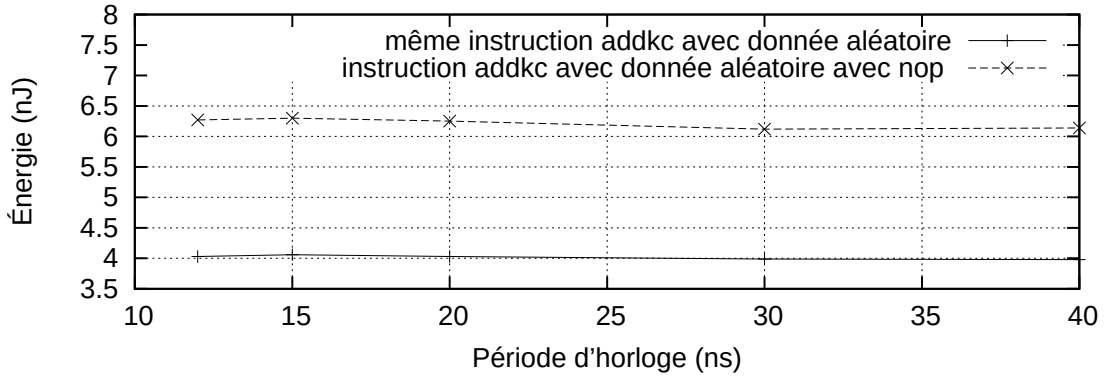


Figure 4.7 Nombre de Joules consommés par l'instruction *addkc* pour différentes fréquences

Cette caractéristique du Microblaze permet de simplifier l'équation finale en éliminant une composante possiblement non-linéaire de l'équation. Le placement et routage pour les différentes équations fait varier la consommation par 3%, ce qui permet de considérer l'effet de la fréquence comme une constante.

En tenant compte des résultats et des simplifications découvertes durant les mesures, il est possible d'écrire l'équation des paramètres technologique du Microblaze comme suit :

$$F(SSP,TP) = \sum_{k=\text{premiere instr}}^{\text{derniere instr}} \sum_{s=0}^{\text{num instr}} M_{sk} l_{sk} l_{sk} \in [0,1]$$

Le coefficient M_{sk} représente la valeur de l'effet du placement et routage sur la consommation de l'instruction sk, la variable l_{sk} sélectionne le facteur dépendant de l'instruction s qui est actif durant le cycle k.

Ceci donne l'équation finale :

$$M(ASP,SSP,TP) = F(SSP,TP)G(ASP,SSP) \tag{4.13a}$$

ce qui donne

$$M(ASP,SSP,TP) = \sum_{k=\substack{\text{première}\\\text{instr}}}^{\substack{\text{dernière}\\\text{instr}}} \sum_{s=0}^{\substack{\text{num of}\\\text{one cycle instr}}} M_{sk} l_{sk} \big((I_{sk} + D_{sk} d_{sk}) l_{sk} + P l_{k\,idle} + M l_{k\,multi} \big) \tag{4.13b}$$

où

$$l_k, l_{idle}, l_{multi}, d_k \in [0,1]$$

4.7 Bus

Le bus est un IP très important, car il constitue la connexion principale entre les différents IP. Le module sélectionné comme bus est l'OPB [57]. Le bus OPB offre la possibilité de connecter de 1 à 16 maîtres et esclaves. Pour le bus OPB, un maître est un IP qui est capable d'initier une communication (processeur) et un esclave est un IP qui ne peut que répondre à une communication (minuterie). Si le bus OPB est connecté à plus d'un maître, il implémente un arbitre qui offre un arbitrage statique (ordre de priorité fixe) ou dynamique. Il est possible d'assigner un maître par défaut si aucune requête n'est effectuée.

Le bus permet des transactions entre deux modules et chacune des transactions comprend une requête, une amorce, un transfert et une terminaison. Puisqu'il n'y a que le transfert (adresse et donnée) qui varie d'une transaction à l'autre, les autres étapes sont considérées comme des constantes spécifiées par l'architecture du bus ou le type de transmission demandée (lecture ou écriture).

4.7.1 Sélection des Paramètres

La particularité du bus est qu'il effectue des transferts de données du maître vers l'esclave ou inversement. Un transfert de donnée est effectué dans une séquence d'action bien précise. Cette séquence est la suivante : la requête, l'amorce, le transfert et la terminaison. L'activité à chacune des étapes varie selon que le transfert, soit une lecture ou une écriture, qu'il y ait un ou plusieurs maîtres ou esclave et s'il y a plusieurs transferts effectués. Mais, seuls le transfert de donnée et l'amorce varient d'une transaction à l'autre car ces deux étapes s'occupent de l'adressage de l'esclave et de la transmission de la donnée. Les autres étapes ne sont que le contrôle de l'arbitre sur le bon fonctionnement de la transmission. L'activité est très similaire ou identique sauf si l'architecture du bus change.

Dans la Figure 4.8, il est possible de voir l'activité typique d'une transaction. Les signaux manquants dans cette image sont le signal RNW (indiquant la lecture ou l'écriture), la donnée et l'adresse. La requête et l'amorce de la communication se font toujours de la même façon. La transmission dure le même temps pour chaque transaction (1 cycle pour l'écriture et 2 cycles pour la lecture) et la terminaison est toujours au second cycle de la transmission. La seule variation entre toutes les transmissions est la valeur du signal RNW, de l'adresse et de la donnée.

La connaissance de ces particularités permet de regrouper les paramètres comme suit :

ASP : lecture ou écriture, changement des bits d'adresse, changement des bits de donnée, transfert simple

SSP : nombre de maîtres, nombre d'esclaves

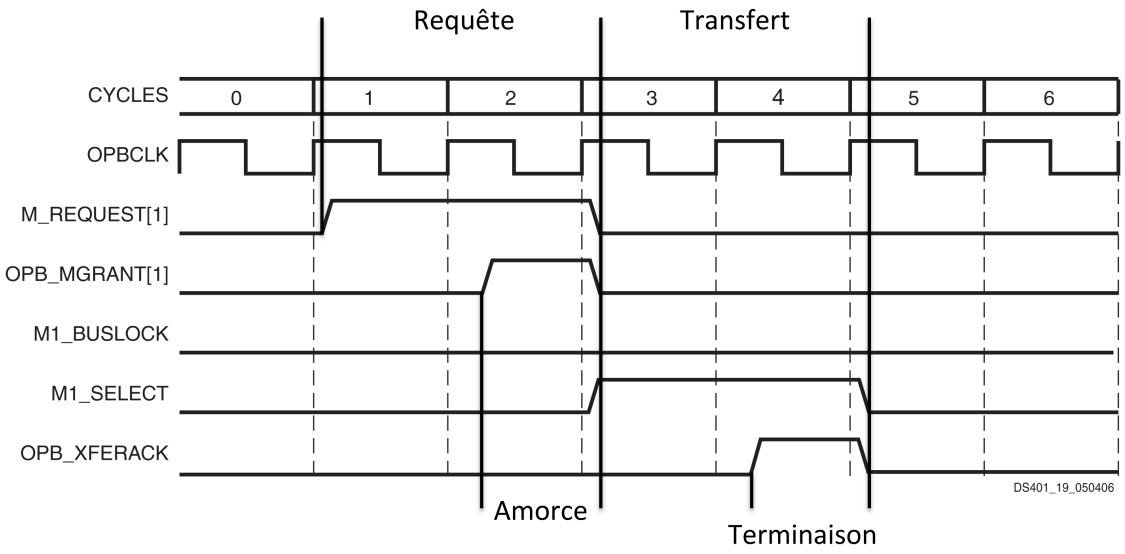


Figure 4.8 Activité typique durant une transaction sur le bus

TP : fréquence d'opération et technologie d'implémentation

Ces paramètres représentent la majorité des bus dans leur ensemble. Certaines particularités des bus doivent être ajoutées s'ils ne peuvent pas être inclus dans les paramètres mentionnés ci-dessus.

4.7.2 Mesure des Paramètres ASP et SSP

Une communication sur le bus OPB est initiée par un maître qui s'est fait octroyer par l'arbitre le droit de communiquer. Tous les maîtres utilisent une interface standard, ce qui donne une impédance identique pour tous les maîtres. Cette particularité permet d'utiliser le Microblaze pour générer l'activité sur le bus. Le Microblaze permet un contrôle de l'activité sur le bus en utilisant le code C. Le code force un accès du processeur sur le bus à une mémoire partagée. Ainsi, il sera possible de faire des lectures et écritures sur le bus à la plage d'adresses voulues. Les accès seront gérés par un script qui génère le code C compilé et exécuté par le Microblaze. Si l'accès se fait dans une plage d'adresse à l'extérieur de celle supportée par la mémoire, il faut changer l'adresse de base de la mémoire pour que l'accès soit valide. Le changement d'adresse se fait par le même script qui manipule le fichier d'architecture du FPGA (MHS).

Durant l'accès d'un maître à un esclave sur le bus, seules l'adresse et la donnée sont analysées. La raison est que tous les signaux de contrôle effectuent la même activité durant une transmission, donc leur effet est interprété comme une constante dans la consommation de puissance. La seule exception est le signal RNW qui indique une lecture ou une écriture.

Pour tenir compte de cette légère différence dans la séquence de la transaction (durant la lecture l'adresse est envoyée. Ensuite vient la donnée, comparativement à l'écriture ou les deux sont envoyés simultanément), la lecture et l'écriture sont analysés séparément.

En plus de la transaction, il faut analyser l'effet de la présence de maître et/ou d'esclave sur le bus durant la communication. Pour cela, le script supervisant les simulations doit modifier l'architecture du système (Spécification Matériel du Processeur (MHS)) pour ajouter ou soustraire des maîtres et esclaves. En plus, il doit modifier le fichier des spécifications logiciel des esclaves et maîtres pour que le processeur soit capable de communiquer avec les différents modules sur le bus.

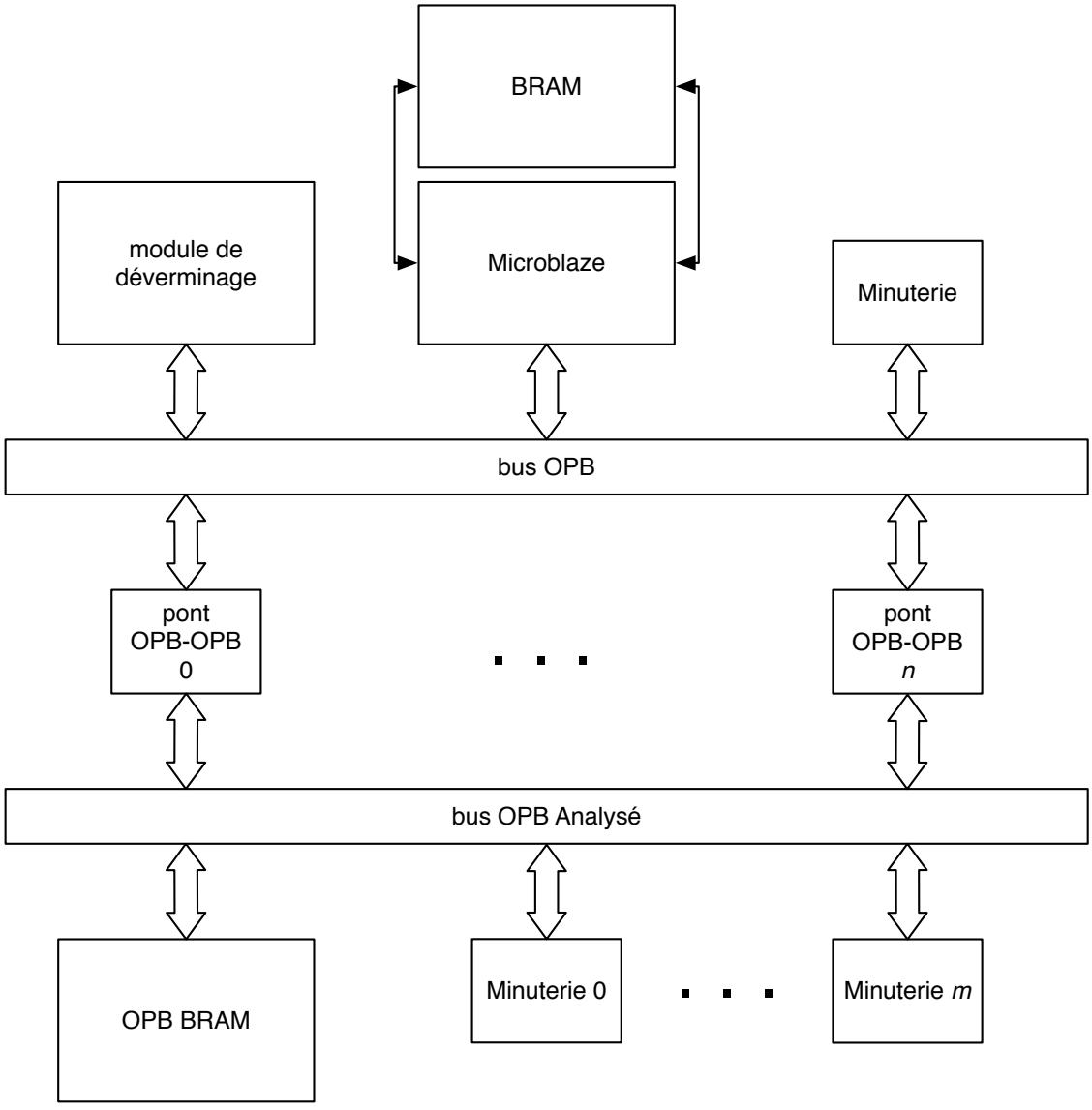


Figure 4.9 Architetcure d'analyse d'un bus

Pour pouvoir évaluer la consommation de puissance du bus avec le maximum de précision, il faut isoler le bus des transactions locales que le processeur effectue. La raison est qu'un processeur vient avec certaines fonctionnalités, comme une minuterie et un module de déverminage, qui peuvent créer des communications involontaires. Puisque le bus doit être analysé avec des maîtres et esclaves, l'utilisation d'un bus secondaire pour l'analyse est nécessaire avec l'utilisation d'un pont entre les deux bus comme élément substitutif à un vrai maître. À cause de l'interface du bus, tous les maîtres ont la même charge capacitive du point de vue du bus. Donc, le pont permet de repsésenter n'importe quel maître connecté au bus OPB. Le pont propage la communication par le processeur sur le second bus. Vu le faible nombre de ressources que requiert le pont, il est possible d'en placer 16 sans que le FPGA soit encombré.

Pour simuler les esclaves supplémentaires, une minuterie est utilisée pour chacun d'eux. Puisqu'une minuterie ne prend pas beaucoup de ressources matérielles, il est possible d'en placer 16 même s'il y a autant de maîtres (voit Figure 4.9). Du point de vue du bus, il n'y a aucune différence entre une minuterie ou tout autre esclave connecté sur le bus.

Le bus tombe dans un cas particulier lorsqu'il y a seulement un seul maître présent. Puisque l'arbitre n'a pas à définir une priorité entre les différents maîtres, lorsqu'il y en a qu'un seul sur le bus celui-ci n'est plus nécessaire et peut être retiré. Ceci permet au maître d'avoir continuellement l'accès au bus. Il faut analyser cette situation particulière séparément car elle cause une non-linéarité dans la consommation de puissance causée par ce changement d'architecture.

La mesure de la consommation d'énergie est obtenue par le calcul de la moyenne de plusieurs accès (entre 100 et 1000). Puisque le bus OPB n'a pas d'activité parasite comme le bus LMB présenté à la Section 4.5.2, seuls les accès causent une activité sur le bus.

4.7.3 Équation Structurelle

La grande simplicité architecturale (un arbitre et un arbre de connexion entre les maîtres et les esclaves) et la répétition des branchements permettent d'obtenir une équation. À cause de l'absence de l'arbitre, s'il n'y a qu'un maître sur le bus, il faut utiliser deux équations. Voici les équations obtenues :

$$G(ASP,SSP) = \left(I_m \frac{ln(M)}{ln(J_m)} + K_m\right)\left(C_m \frac{ln(S)}{ln(D_m)} + E_m\right)\left((A_{pente}Bit_{addr} + A_{const})Bit_{data}\right) + \left(L_m \frac{ln(M)}{ln(N_m)} + O_m\right)\left(F_m \frac{ln(S)}{ln(H_m)} + P_m\right)(B_{pente}Bit_{addr} + B_{const}) \quad (4.14)$$

avec

$$M = m \in [1,16]$$

$$C_m,\ldots,P_m = \begin{cases} \text{valeurs pour un maitre} & \text{si } m = 1, \\ \text{valeurs pour plusieurs maitres} & \text{si } m > 1 \end{cases}$$

$C_m,\ldots,P_m$ sont les constantes de l'équation. Les paramètres J_m et N_m représentent l'ordre du logarithme qui représente l'impact de l'ajout d'un maître sur la consommation, D_m et H_m pour les esclaves. Les paramètres I_m, K_m, L_m et O_m changent l'amplitude de cet impact et la valeur de base de l'impact selon le nombre de maître. Les paramètres C_m, E_m, F_m et P_m font la même chose pour les esclaves. Ces constantes sont définies pour le cas où le bus contient un ou plusieurs maîtres. La forme de l'équation reste la même, par contre les valeurs des paramètres changent selon le nombre de maître (à cause de la présence de l'arbitre). Les paramètres $A_{pente}, A_{const}, B_{pente}etB_{const}$ donnent la valeur de la consommation de base du bus sans module connecté. B_{pente} et B_{const} représentent l'impact sur la consommation de puissance du nombre de bits d'adresse en transition. A_{pente} et A_{const} représentent l'impact du nombre de bits d'adresse en transition par rapport au nombre de bits de donnée en transition. M représente le nombre de maîtres présents sur le bus, S représente le nombre d'esclaves. Bit_{addr} et $Bitdata$ représentent le nombre de bits à 1 durant la transmission. Quand un maître ou un esclave ne communique pas, il doit mettre à zéro ses ports d'adresses et de données. Ainsi, s'il n'y a pas de communication l'adresse et la donnée dans le bus sont zéro.

L'analyse de l'activité du bus OPB commence par la requête. Une fois la requête détectée, le système comptabilise la transition des signaux de contrôle, de l'adresse et de la donnée. Cette analyse perdure tant que le signal d'acquiescement («acknowledge») de l'esclave et les signaux de transfert ne sont pas redescendus. Puisque les ponts OPB-OPB s'activent seulement si l'adresse de transfert est dans leur plage valide, il est possible d'envoyer une requête par n'importe quel pont.

4.7.4 Mesure des Paramètres TP et Équations Finales

Le même patron d'activité est utilisé pour la mesure des paramètres technologique. Dans ce cas, le nombre de points de mesure (nombre de bits) est diminué pour minimiser le temps de l'analyse. La valeur mesurée, avec et sans arbitre,s est utilisée pour définir l'équation représentant l'effet de la fréquence sur la consommation d'énergie.

Le bus est un module très régulier et composé surtout d'interconnexion entre les différents modules communiquant sur le bus. Donc, l'effet de la fréquence est très simple. L'effet sera plus grand sur les contraintes de routage imposé sur les interconnexions du bus avec les différents maîtres et esclave. Ceci se voit immédiatement avec des résultats qui sont identiques. Puisque la période minimale est de 12 ns soit $83,\overline{3}$MHz, le bus OPB ne semble pas être affecté par le changement de fréquence. Seul, un changement négligeable des résultats est observé. Suivant cette observation $F(SSP, TP)$ ne fera qu'ajuster l'équation structurelle 4.7.3 dans l'équation finale. Voici le format de l'équation des paramètres technologiques :

$$F(SSP, TP) = AP + B \tag{4.15}$$

où, P représente la période en nanoseconde, A et B sont des constantes sélectionnées pour corriger la valeur obtenue par l'équation $G(ASP, SSP)$ présenté dans l'équation 4.14 pour tenir compte des délais temporels dans les résultats.

L'équation finale devient alors :

$$M(ASP, SSP, TP) = F(SSP, TP)G(ASP, SSP) \tag{4.16a}$$

ce qui donne

$$\begin{aligned} M(ASP, SSP, TP) = (AP + B) & \\ \left(\left(I_m \frac{ln(M)}{ln(J_m)} + K_m\right)\left(C_m \frac{ln(S)}{ln(D_m)} + E_m\right)\left((A_{pente} Bit_{adr} + A_{const}) Bit_{data}\right)\right. & \\ \left. + \left(L_m \frac{ln(M)}{ln(N_m)} + O_m\right)\left(F_m \frac{ln(S)}{ln(G_m)} + H_m\right)\left(B_{pente} Bit_{addr} + B_{const}\right)\right) & \end{aligned} \tag{4.16b}$$

avec

$$M = m \in [1, 16]$$

$$C_m, \ldots, O_m = \begin{cases} \text{valeurs pour un maître} & \text{si } m = 1, \\ \text{valeurs pour plusieurs maîtres} & \text{si } m > 1 \end{cases}$$

CHAPITRE 5

RÉSULTATS THÉORIQUES ET EXPÉRIMENTAUX

Cette section détaille les résultats obtenus à la Section 4. Comme démontré, les modèles de consommation ont des performances en fonction de la qualité des mesures, de la sélection des paramètres et de la formation des équations. Dans cette section, l'analyse de la qualité des estimations et des modèles permettent de comprendre les résultats obtenus et de déterminer si les modèles permettent de déterminer précisément la consommation de puissance.

L'analyse a été effectuée en utilisant l'outil Xilinx EDK pour générer les systèmes. Par la suite, le simulateur utilisé est Modelsim 6.3c de Mentor. L'analyse de l'activité pour obtenir la consommation de puissance de référence est effectuée par Xpower de Xilinx (l'outil vient avec la suite EDK et ISE de Xilinx). La mesure de la précision des modèles est effectuée avec le script Perl mentionné à la Section 4.2. Le script effectu la comparaison des résultats pour fournir le taux d'erreur. La précision de Xpower a été souvent discutée dans la littérature. Quelques articles mentionnent que l'erreur de Xpower est de 40% et possiblement plus [23][25]. Par contre, un grand nombre d'articles utilisent les résultats de Xpower [46][14][4] et offre des résultats très précis. Avec les années, l'outil Xpower s'est amélioré et permet aujourd'hui d'offrir une excellente estimation de la consommation. Les mesures de l'article de Jingzhao Ou et Viktor K. Prasanna [37] montrent que l'erreur moyenne d'estimation de Xpower est de 6.8%.

Pour générer l'activité dans le système, des bancs de test exécutés par le processeur sont utilisés. Les bancs de test utilisent le processeur comme générateur d'activité pour les différents IP comme la minuterie. La minuterie et le PIC sont utilisés pour générer des interruptions. Ceci ne fait qu'interrompre le banc de test temporairement et permet de tester les deux IP et effectue de l'activité pour le processeur et le bus. Les deux bancs de tests sont le Dhrystone [48] et le Coremark [13]. La Figure 5.1 montre la structure du système de test utilisée pour valider les différents IP.

Les informations extraites des simulations Modelsims correspondent aux informations disponibles par une simulation TLM [9] de l'outil Space Codesign [5, 7, 32, 33]. Puisque les modèles sont utilisés dans cet outil, il faut s'assurer que l'information utilisée par les modèles soit identique à celle que l'on obtient par l'outil Space CoDesign.

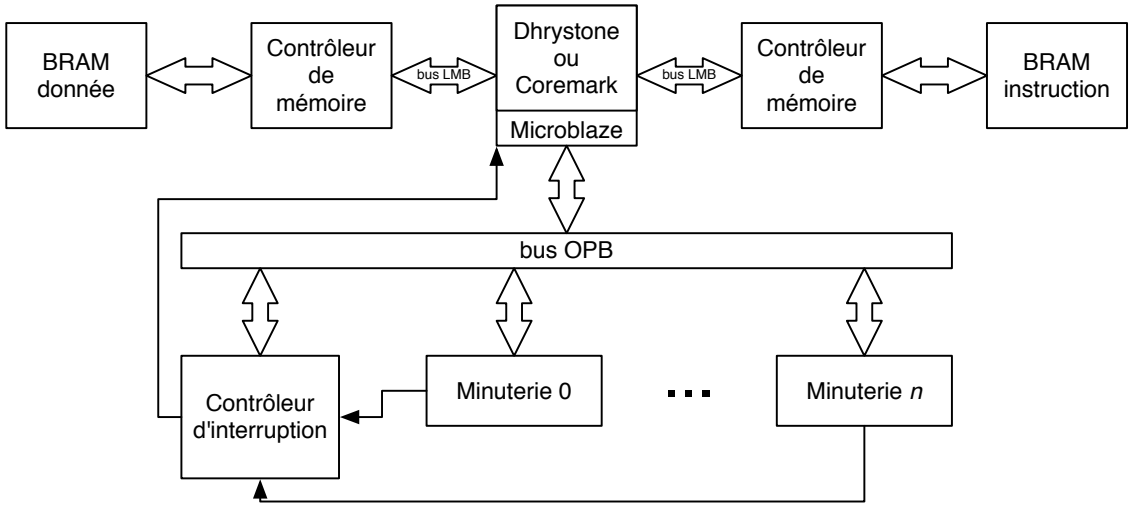


Figure 5.1 Architecture du système de test

5.1 Minuterie

La consommation de la minuterie est très constante, peu importe la fréquence de fonctionnement. Ceci est dû au fait que la minuterie est composé de modules très simples. Outre l'interface OPB, elle ne contient que des registres de décompte et des registres de contrôle avec une logique de contrôle assez simple.

La simplicité du circuit se voit dans la consommation d'énergie constante de la Figure 5.2. Il est possible de voir que la consommation d'énergie pour un même nombre de cycles de décompte est la même, peu importe la fréquence. Ceci indique que l'architecture de la minuterie n'est pas affectée par une contrainte de fréquence plus sévère. La raison de cette particularité est dûe à la minuterie. Elle est composé d'un circuit très simple qui n'est pas implémenté différemment à cause des contraintes de délai.

En changeant la période de décompte, il est possible de voir que la consommation se comporte de façon linéaire. Puisque la consommation est affectée par la proportion de cycle passé en décompte par rapport à une transaction sur le bus OPB, chaque cycle ajoute un taux d'activité très similaire. Ceci permet d'obtenir une droite qui est déterminée par une constante représentant la consommation de la communication du bus.

Puisque la consommation d'énergie est très stable d'une fréquence à l'autre, le modèle donne des résultats très précis. Ceci est visible avec les Figure 5.3 et Figure 5.4 qui représentent la consommation de la minuterie au cours de l'utilisation des bancs de tests.

Puisque la minuterie n'est pas un circuit très complexe, la consommation d'énergie n'est pas très élevée, et est très linéaire. L'estimation du modèle est très précise par rapport au nombre de variables requises pour effectuer l'estimation. Par contre, pour une activité légè-

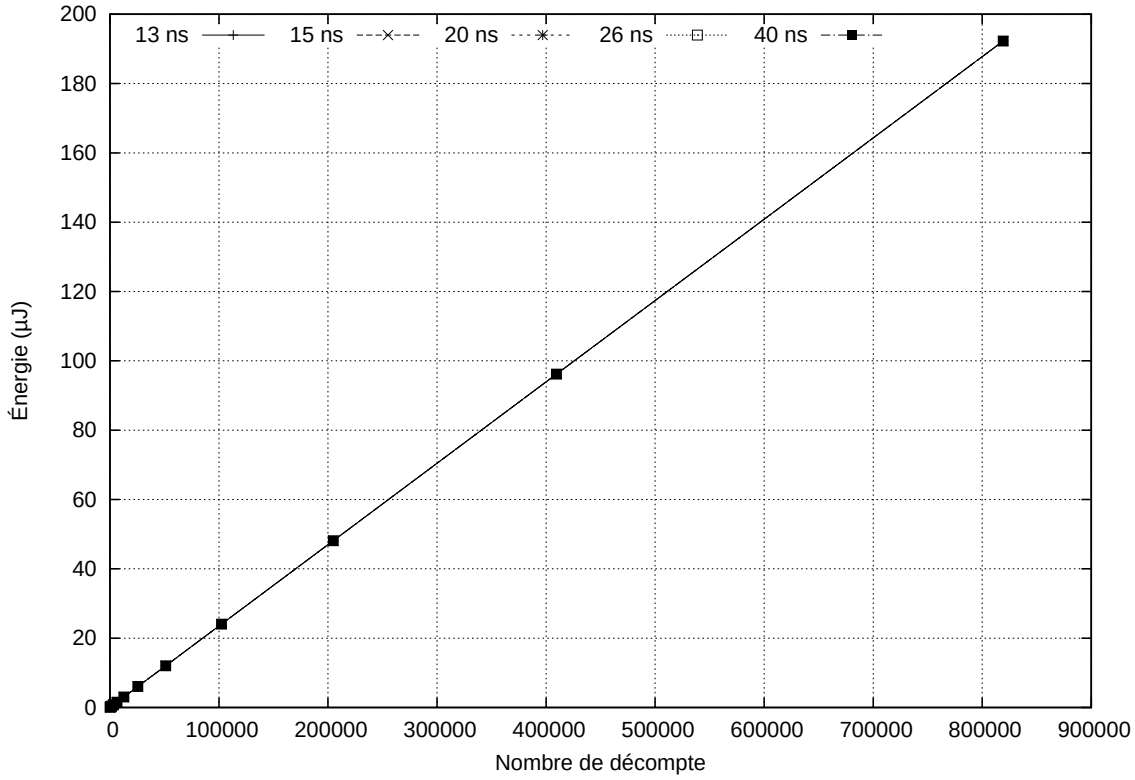


Figure 5.2 Consommation d'énergie de la minuterie

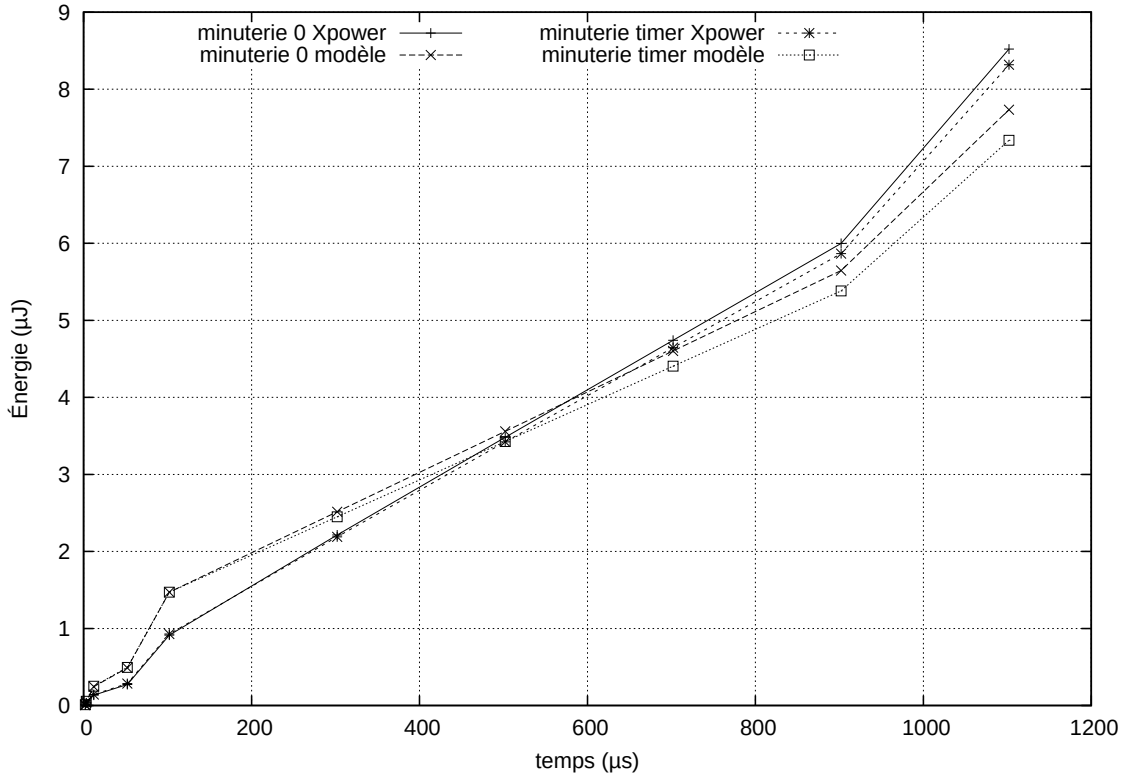


Figure 5.3 Minuterie utilisée avec le Dhrystone à 12 ns

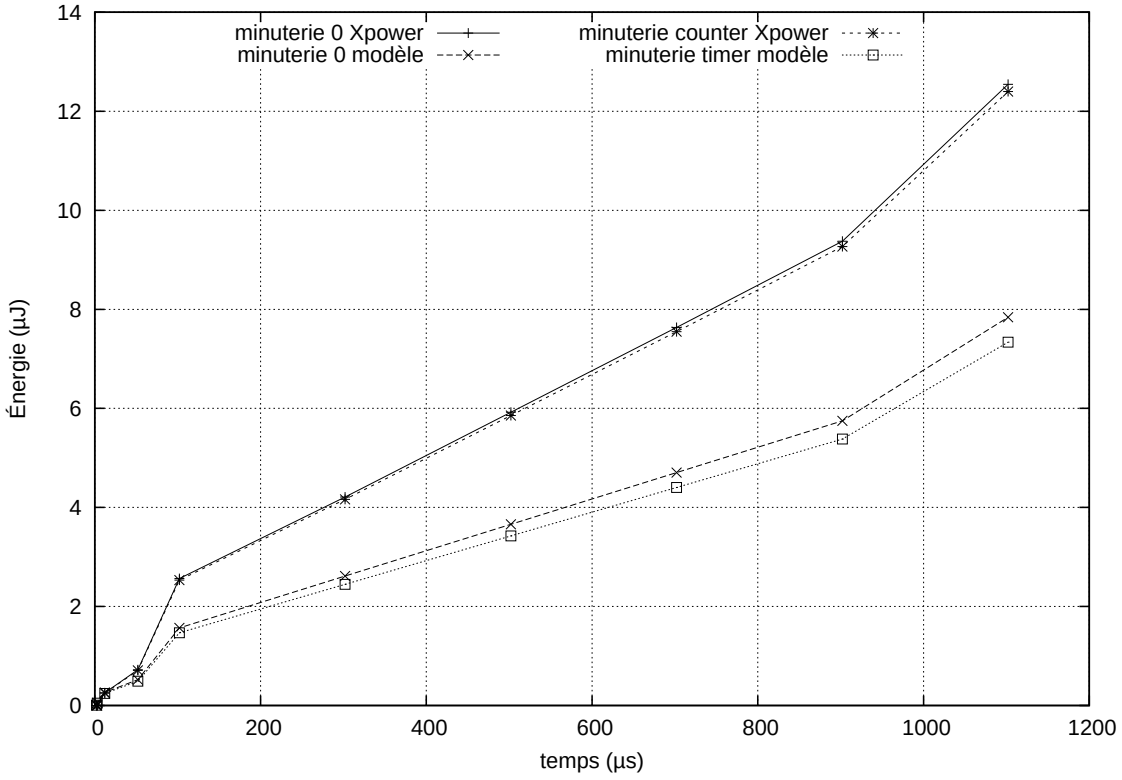


Figure 5.4 Minuterie utilisée avec le Coremark à 12 ns

rement plus élevée à la Figure 5.4, la valeur de la consommation de puissance de la minuterie durant l'utilisation du Coremark est plus élevée que l'estimation. Ce résultat indique que le modèle ne tient pas compte de tous les paramètres d'activités du compteur ou que Xpower effectue des estimations qui contiennent des erreurs. Pour le Virtex2, l'outil Xpower indique que l'état de leurs outils d'estimation a une erreur de 20%. Il est possible que le modèle soit plus proche du vrai comportement que précédemment pensé, car si on tient compte de l'erreur de Xpower, elle n'est qu'à 20% par rapport à Xpower. Il est probable que le PAR soit une partie de la cause d'erreur. Puisque le Dhrystone n'a pas la même erreur, Xpower peut avoir analysé une activité qui n'est pas dans le modèle. Bien que le résultat absolu pour la Figure 5.4 ne soit pas bon, la corrélation entre les deux courbes est de 0,8 (pour les deux minuterie). Ceci indique que le modèle représente bien l'importance relative de la consommation à cause du comportement similaire des deux courbes. Donc, il ne s'agit que d'ajuster l'équation avec un facteur, ce qui supporte l'hypothèse du PAR différent.

5.2 Contrôleur d'Interruptions de Processeur

Comme détaillé à la Section 4.4.2, le PIC est un IP très simple et très linéaire dans son comportement et son architecture. Ceci se perçoit à la Figure 5.5 qui montre une consommation linéaire du PIC avec l'ajout de lignes d'interruption.

Avec cette grande linéarité dans la consommation d'énergie de l'IP, il est possible de facilement vérifier le comportement de la consommation d'énergie du module à différentes fréquences. En observant la Figure 5.6, il est possible de voir que la consommation est stable, peu importe la fréquence , et continue à être linéaire par rapport au nombre de lignes d'interruption.

Cette linéarité de la consommation par rapport au nombre de lignes d'interruption, et la consommation d'énergie stable par rapport à la fréquence, permettent au modèle de rester fidèle à la consommation réelle. L'estimation reste précise durant l'utilisation des bancs de tests (voir les Figure 5.7 et Figure 5.8).

Le modèle d'estimation du PIC donne un résultat très différent de Xpower. La raison est probablement dûe au placement et routage du PIC, qui est très différent du PAR original des mesures. Bien que l'estimation suit les mêmes variations de croissance (les pentes sont similaires même avec le point de flexion), l'erreur de l'estimation est très grande. Le circuit du PIC comporte peu de contraintes de performance car il ne fait que répéter les interruptions reçues. Ceci permet au synthétiseur d'éparpiller le circuit du PIC au travers du FPGA pour permettre au bus, mémoire et processeur d'être le plus compact possible. Ceci cause de longue connections au PIC ; ce qui, en retour, augmente la consommation par rapport aux mesures

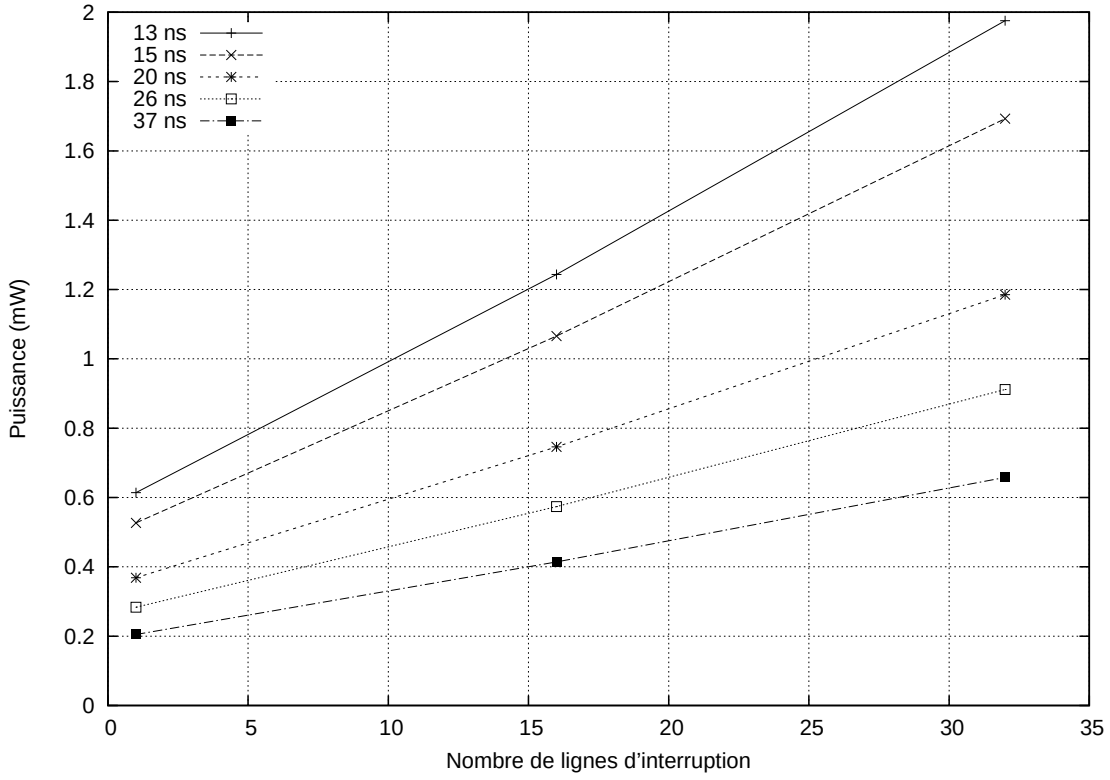


Figure 5.5 Mesure de consommation du PIC pour dix mille interruptions

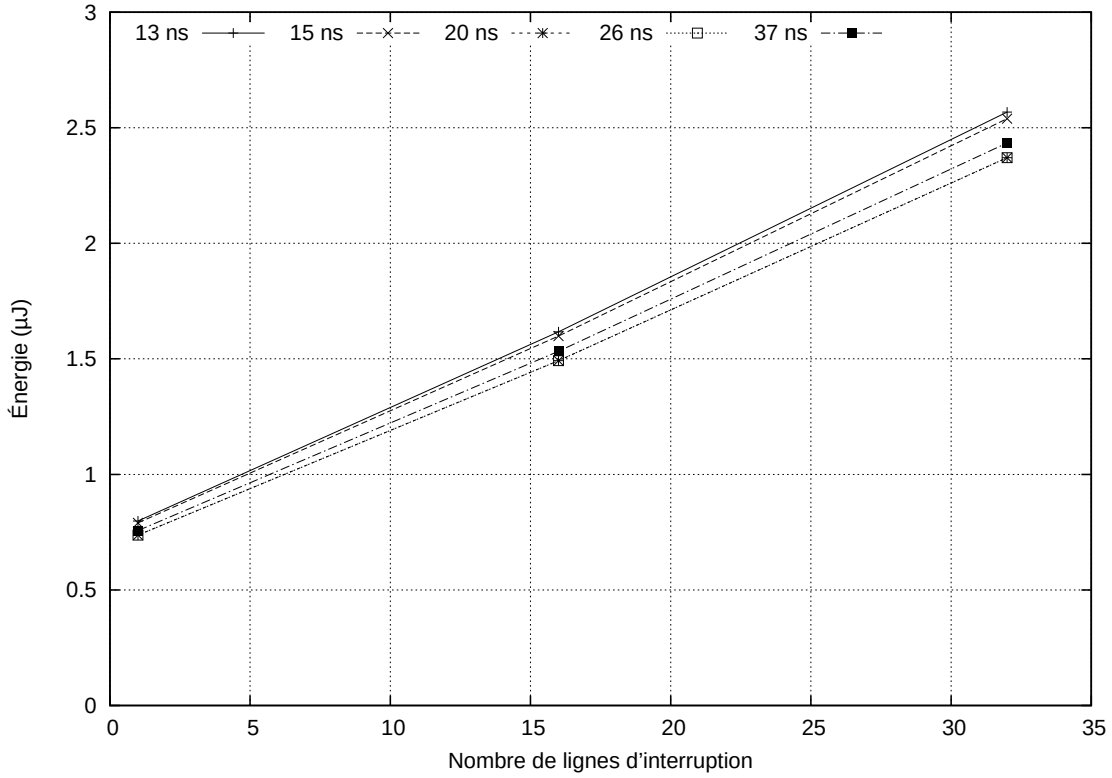


Figure 5.6 Consommation du PIC à différentes fréquences pour dix mille interruptions

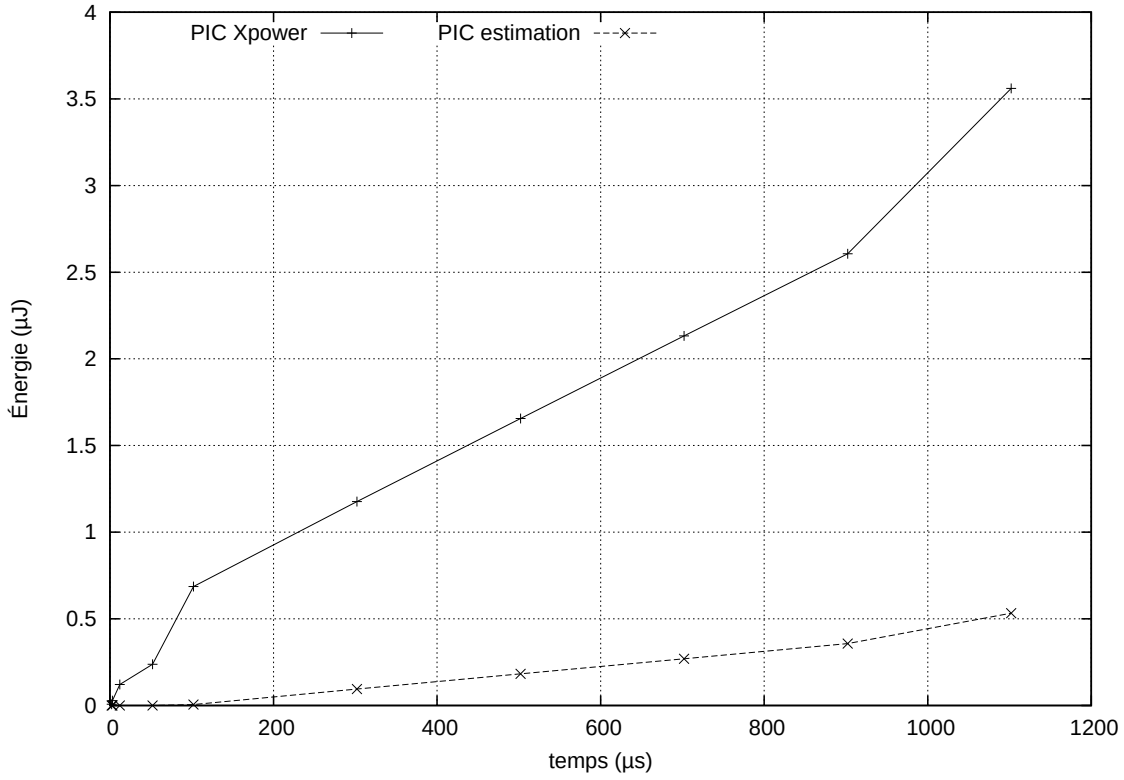


Figure 5.7 PIC utilisé avec le Dhrystone à 12 ns

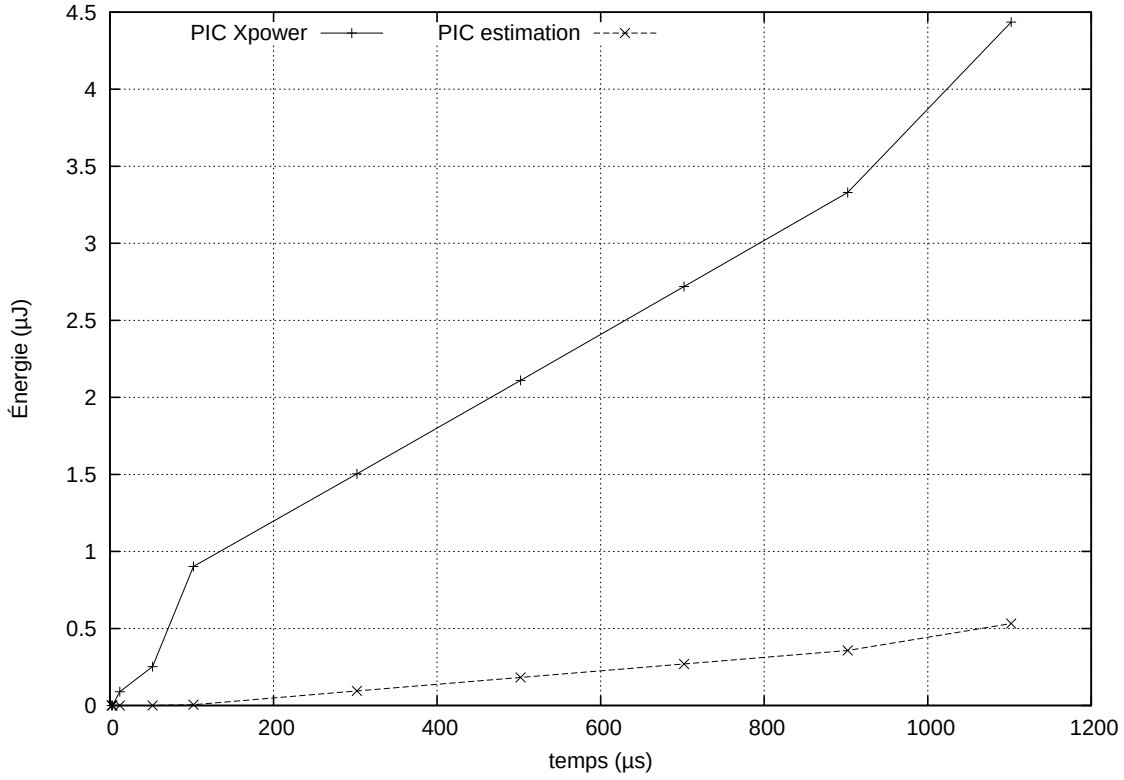


Figure 5.8 PIC utilisé avec le Coremark à 12 ns

effectuées avec un circuit minimal. Même en tenant compte de l'erreur de Xpower, l'erreur reste très grande. Ce cas démontre l'importance du PAR dans la consommation car il cause une estimation avec une erreur de presqu'un ordre de grandeur. Ici encore, la corrélation entre l'estimation du modèle et la mesure de Xpower est de 0,9831 pour le Dhrystone et 0,9997 pour le Coremark. Ce comportement quasi identique montre que le modèle représente bien la relation de la consommation entre les différentes activités du PIC. Par contre, le PAR change l'amplitude de cette consommation à cause de la faible contrainte sur ce dernier. Ceci a permis au synthétiseur de changer grandement le placement et routage des ressources du PIC.

5.3 Mémoire

L'estimation de la consommation de puissance de la mémoire est un problème plus complexe que le PIC et la minuterie. Dans la Section 4.5.2, il a été discuté que l'implémentation de la communication entre la mémoire et le processeur a un comportement particulier. Le processeur envoie sa commande de lecture et d'écriture de la même façon : il envoie une adresse et la valeur contenue dans le registre qui est requis par la commande assembleur. Ensuite, la mémoire renvoie la valeur qui est contenue à l'adresse spécifiée par le processeur. Ce qui distingue une écriture d'une lecture est le signal RNW. Le contrôleur de mémoire utilise ce signal pour activer le mode lecture ou écriture de la mémoire et le processeur ignore la valeur retournée par la mémoire s'il est en mode écriture.

Dans le cas de l'écriture, la consommation peut être connue car, à l'envoi, l'adresse et la donnée sont connues. Puisque la valeur retournée par la mémoire est la même qui a été écrite, le modèle connaît toute l'activité qui a été effectuée. Dans ce cas, il est possible d'estimer la consommation d'énergie précisément. Par contre, dans le cas de la lecture il n'est pas garanti que le modèle estime la consommation précisément. Normalement, le modèle de simulation TLM du processeur fournit seulement l'adresse qui est lue. Le modèle de la mémoire fournira la valeur lue à cette adresse. Le Microblaze envoie la valeur contenue dans le registre à la mémoire. Puisque cette activité n'est pas utile, elle n'est donc pas connue durant une simulation TLM. Il manque une partie de l'activité qui affecte la mémoire, car l'implémentation de cette dernière et du Microblaze ne se conforme pas au format d'une transaction TLM effectué dans Space Codesign. La précision de l'estimation en sera automatiquement affectée.

À cela, il faut aussi ajouter les cycles où le Microblaze n'accède pas à la mémoire (les cycles morts). Le Microblaze contient un bus interne pour connecter tous les ports de données externes comme illustré à la Figure 5.9. L'implémentation est effectuée de façon que le bus LMB ne soit pas découplé des autres bus. Donc, quand le Microblaze envoie des données sur

le bus IF qui est destiné pour une autre interface que le bus LMB, ces données se retrouvent néanmoins sur le bus LMB bien que les signaux de contrôle ne soient pas activés. Même s'il n'y ait pas d'accès à la mémoire, il y a une activité car la mémoire se comporte comme en lecture (ce qui est son comportement par défaut).

Cette activité est dépendante du code exécuté. Cela requiert que cette activité soit fournie par le modèle du processeur, ce qui n'est pas possible en respectant les modèles TLM de simulation. Car si un IP n'est pas utilisé activement dans le système, il n'est pas simulé pour sauver du temps de simulation. Dans ce cas, une estimation précise sera difficile à obtenir. L'approximation prévilégié utilise les valeurs écrites et lues à la mémoire pour estimer celle qui sera présente durant les cycles morts. La supposition est que les données transitées avec la mémoire ont un lien avec celle qui se retrouve sur le bus interne durant les cycles morts du bus LMB. Donc, elles auraient un taux d'activité similaire.

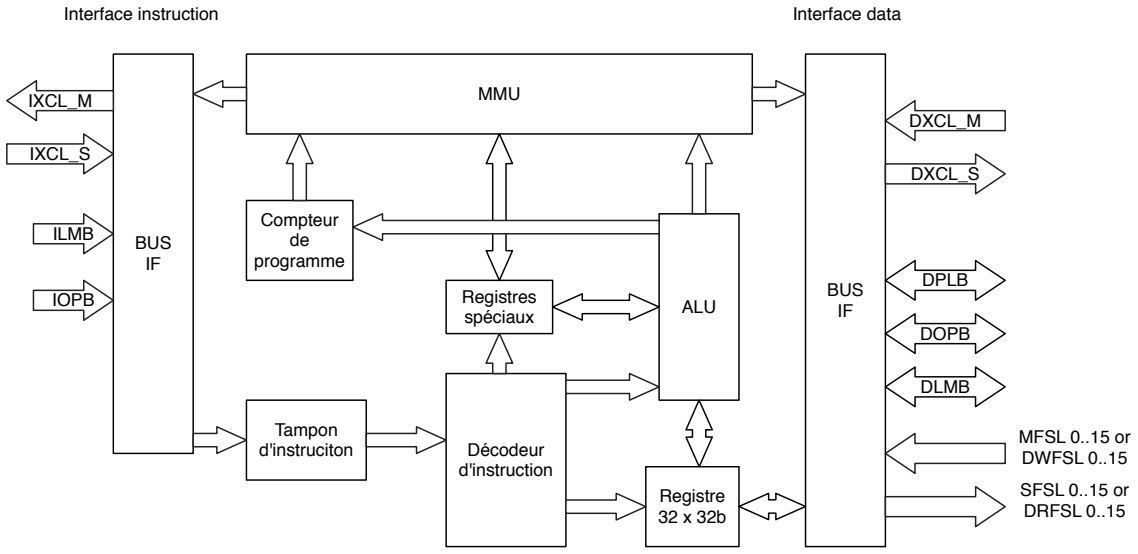


Figure 5.9 Architetcure interne du Microblaze

En regardant les résultats obtenus dans les simulations avec le code du Dhrystone (Figure 5.10, Figure 5.11, Figure 5.12 et Figure 5.13) et de Coremark (Figure 5.14 et Figure 5.15), il est possible de voir que les résultats varient énormément. Dans tous les cas, quand la proportion de cycles morts diminuent, il est possible de voir que la précision augmente (l'augmentation est soit drastique ou de quelque points de pourcentages). De plus, ce phénomène se répète quand on compare la mémoire d'instruction et de donnée. Puisque la mémoire d'instruction a très peu de cycle mort, sa précision est plus élevée. De plus, le Microblaze ne fait que lire de la mémoire d'instruction, alors il n'envoie jamais de donnée à la mémoire. Ce qui permet de connaitre l'activité uniquement en observant la lecture dans sa définition traditionnelle. Ici nous obtenons une erreur maximale qui peut aller jusqu'à

35% pour la mémoire de donnée, ce qui n'est pas idéal. Par contre, l'erreur maximale de l'estimation de la mémoire d'instruction est de 8% (pour un cas, l'erreur tombe à 1% sinon).

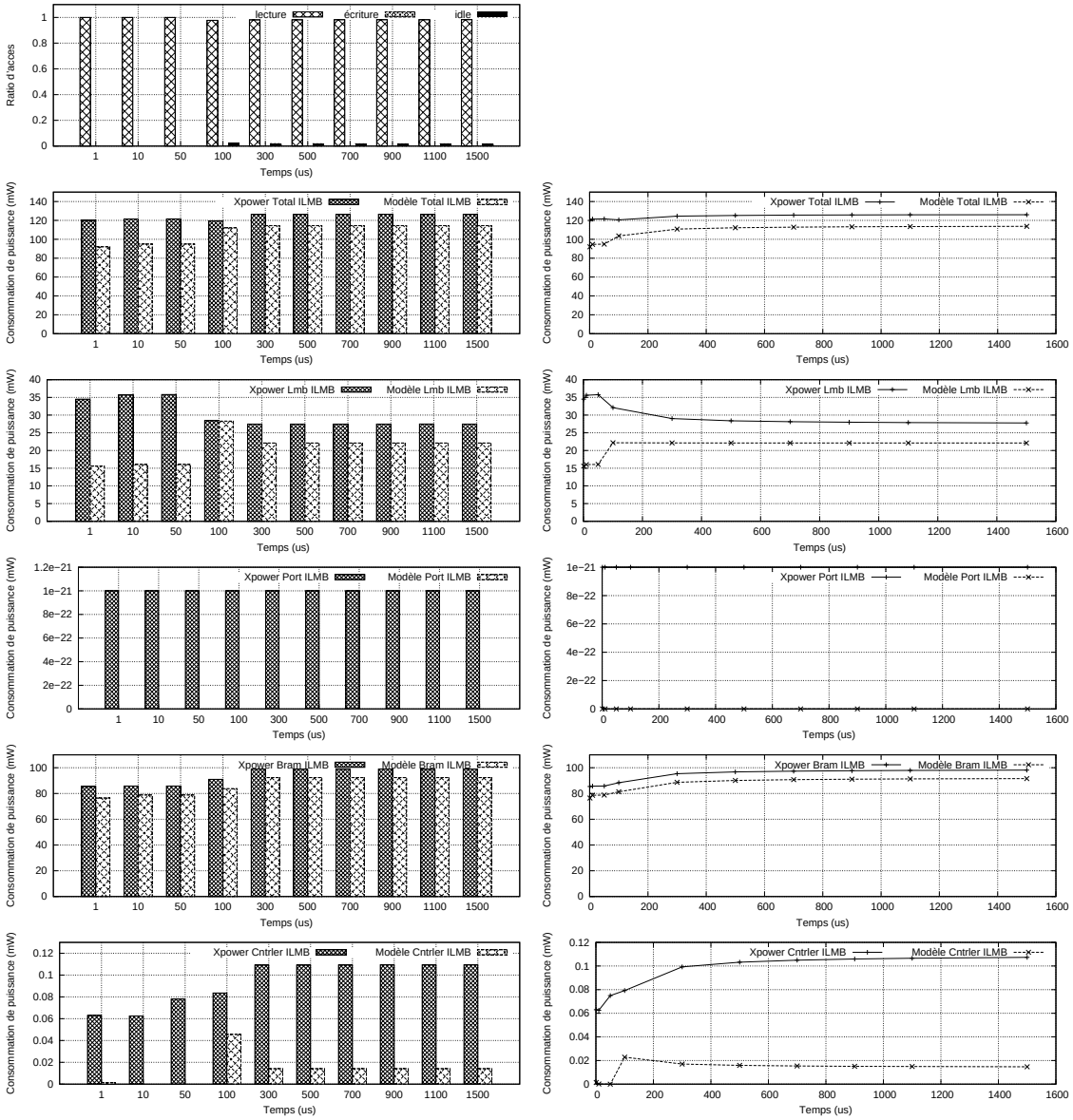


Figure 5.10 Graphes de la consommation de la mémoire avec le Dhrystone et une mémoire de 16kb, coté ILMB à 12 ns

La variation dans l'erreur indique que le modèle comporte des limitations dans sa capacité à représenter l'activité et la consommation. Pour résoudre le problème, il faut avoir accès à l'activité sur le bus interne au Microblaze durant les cycles mort de la mémoire et aussi avoir accès à l'information contenue dans les registres durant une simulation TLM. Une autre solution est de modifier l'interface LMB pour bloquer l'activité parasitique. Dans le cas de

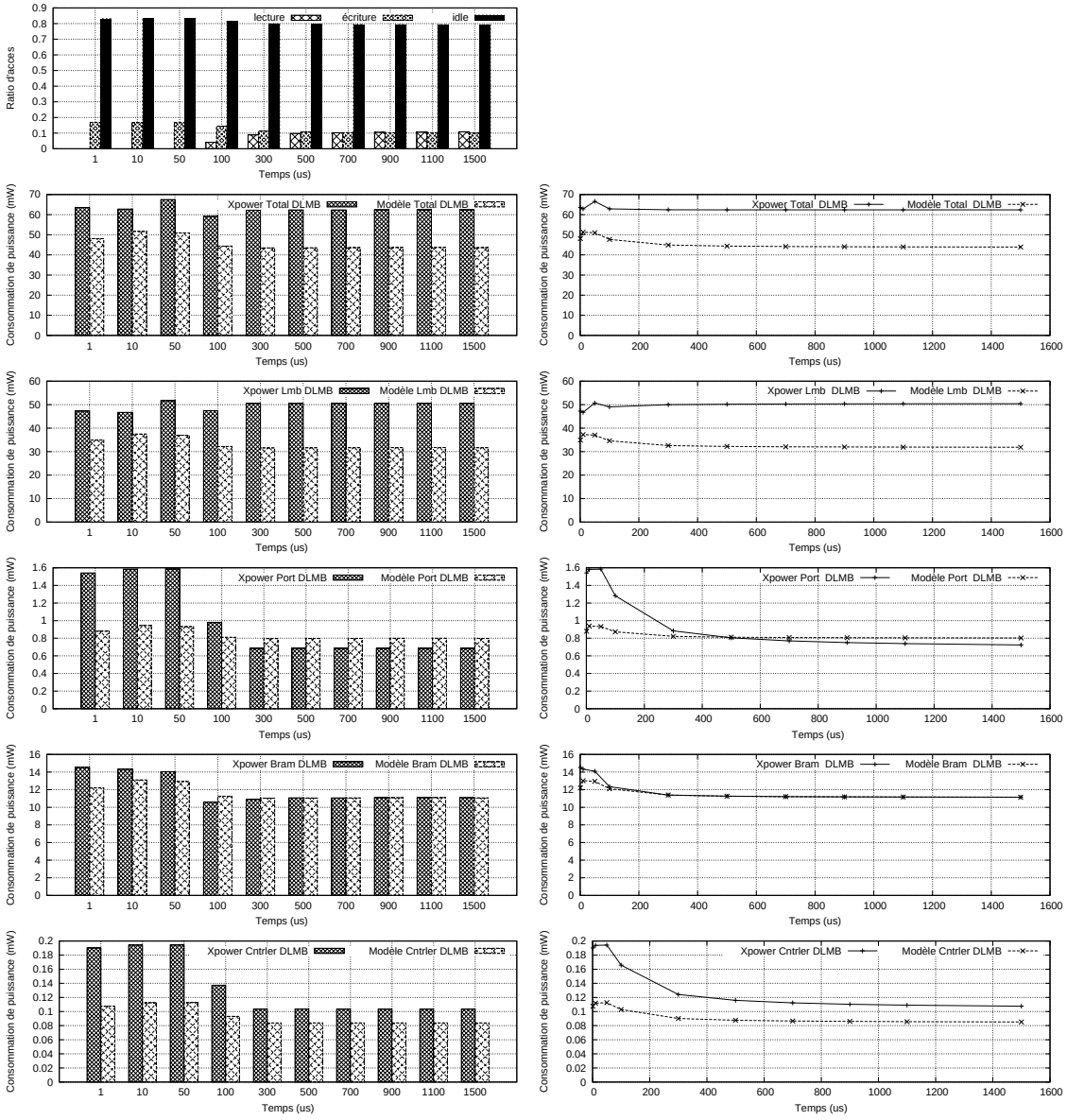


Figure 5.11 Graphes de la consommation de la mémoire avec le Dhrystone et une mémoire de 16kb, coté DLMB à 12 ns

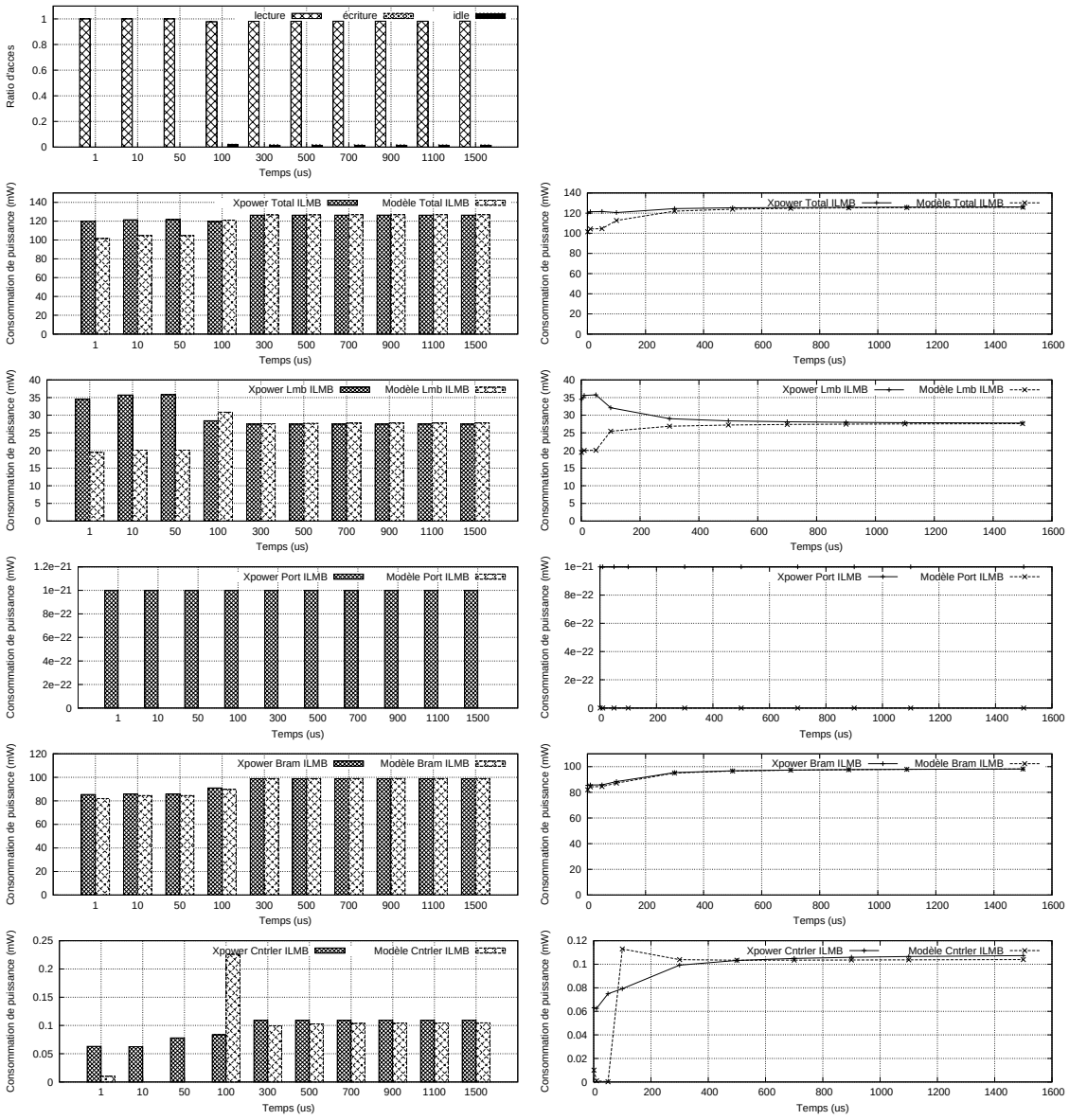


Figure 5.12 Graphes de la consommation de la mémoire avec le Dhrystone et deux mémoires. Instruction de 8kb et donnée de 8kb, coté ilmb à 12 ns

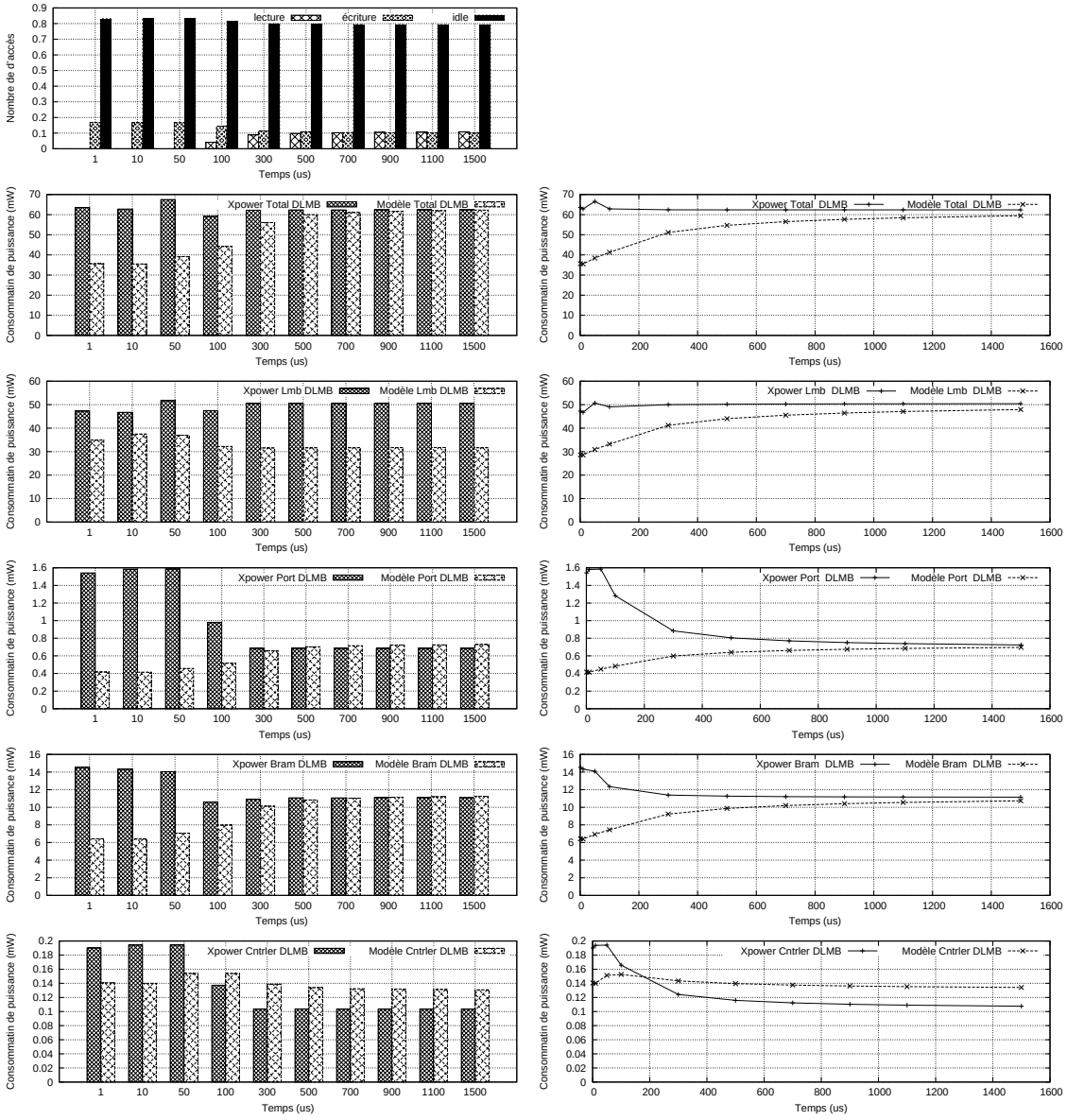


Figure 5.13 Graphes de la consommation de la mémoire avec le Dhrystone et deux mémoires. Instruction de 8kb et donnée de 8kb, coté dlmb à 12 ns

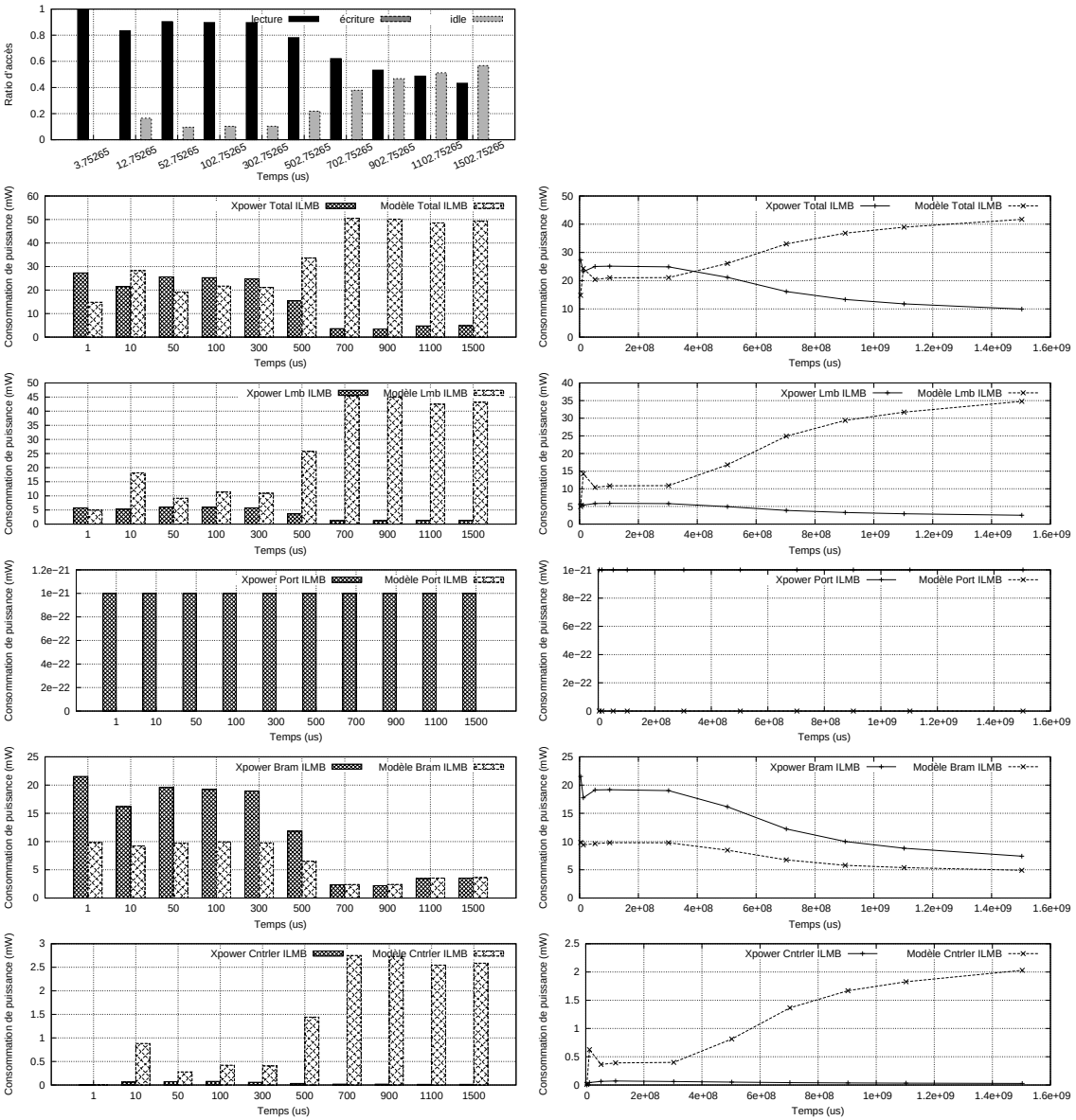


Figure 5.14 Graphes de la consommation de la mémoire avec le Coremark et une mémoire de 64kb total à 12 ns coté ILMB

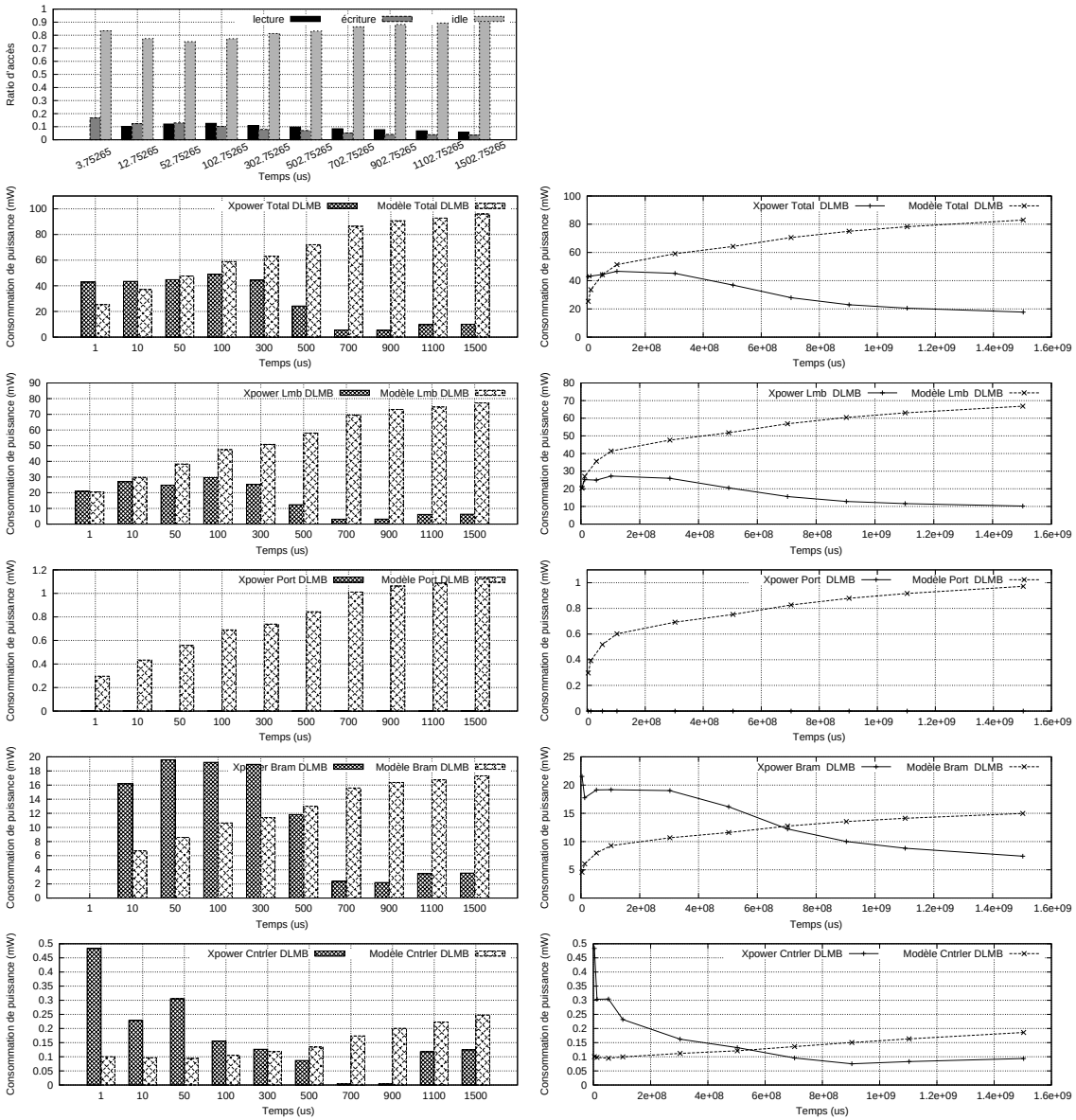


Figure 5.15 Graphes de la consommation de la mémoire avec le Coremark et une mémoire de 64kb total à 12 ns coté DLMB

la mémoire de donnée, l'erreur la plus importante se situe surtout dans l'estimation du bus LMB et l'estimation du contrôleur. La raison est que les deux modules sont majoritairement des interconnexions avec peu de logique. L'adresse varie plus souvent que la donnée durant les cycles morts. La donnée charge et décharge toute la BRAM ce qui cause une grande consommation. L'adresse ne fait que sélectionner les registres ou la sortie à activer, l'erreur impliquée est plus petite. Par contre, le bus LMB et le contrôleur traitent l'adresse et la donnée de la même façon (c.-à.-d. des connexions avec peu de logique). Ceci fait que l'activité de l'adresse et de la donnée est similaire vis-à-vis de leur impact sur la consommation de ces modules. Dans tous les cas, il est à noter que plus la proportion de cycle mort (idle) est grande plus l'erreur d'estimation est grande. La mémoire de donnée contient une plus grande erreur et, ce pour toutes les mémoires, si la proportion de cycle mort augmente l'erreur augmente autant.

5.4 Processeur

Le modèle du processeur fournit des estimations de consommation d'énergie basées sur l'instruction assembleur qui est exécutée. Ceci permet au modèle de rester précis même si l'estimation d'une instruction ne l'est pas. Par contre, il faut que toutes les instructions soient précises pour que le modèle soit performant. Puisque le processeur ne contient pas énormément d'option, ceci fait que la consommation du processeur est assez stable entre les différentes architectures (voir la Figure 4.7). L'analyse de la consommation du Microblaze lors de l'utilisation du Dhrystone donne d'excellents résultats à la Figure 5.16.

La possibilité de compartimenter la consommation du processeur en différentes instructions permet d'obtenir un modèle précis plus facilement. Il est possible de voir que l'estimation est plus faible au début de la simulation et légèrement plus élevée vers la fin. La raison est que les mêmes instructions sont appelées continuellement durant l'initialisation du processeur. La sommation de l'erreur des instructions est différente de zéro et cette erreur s'accumule durant un grand nombre de cycles. L'erreur d'une instruction est plus grande que l'erreur totale des multiples instructions exécutées dans le temps, puisque l'erreur passe du positif au négatif ou vice-versa continuellement (ce qui annule l'erreur à long terme). Malgré cela, l'estimation reste précise à 5%. De plus, il est possible de voir que la consommation est proportionnelle au nombre d'instructions exécutées durant une période de temps fixe. Ceci indique que l'ALU cause une grande part de la consommation du processeur. Ceci est compréhensible car le reste du processeur est composé de registres ou de circuits relativement simples comme un décodeur.

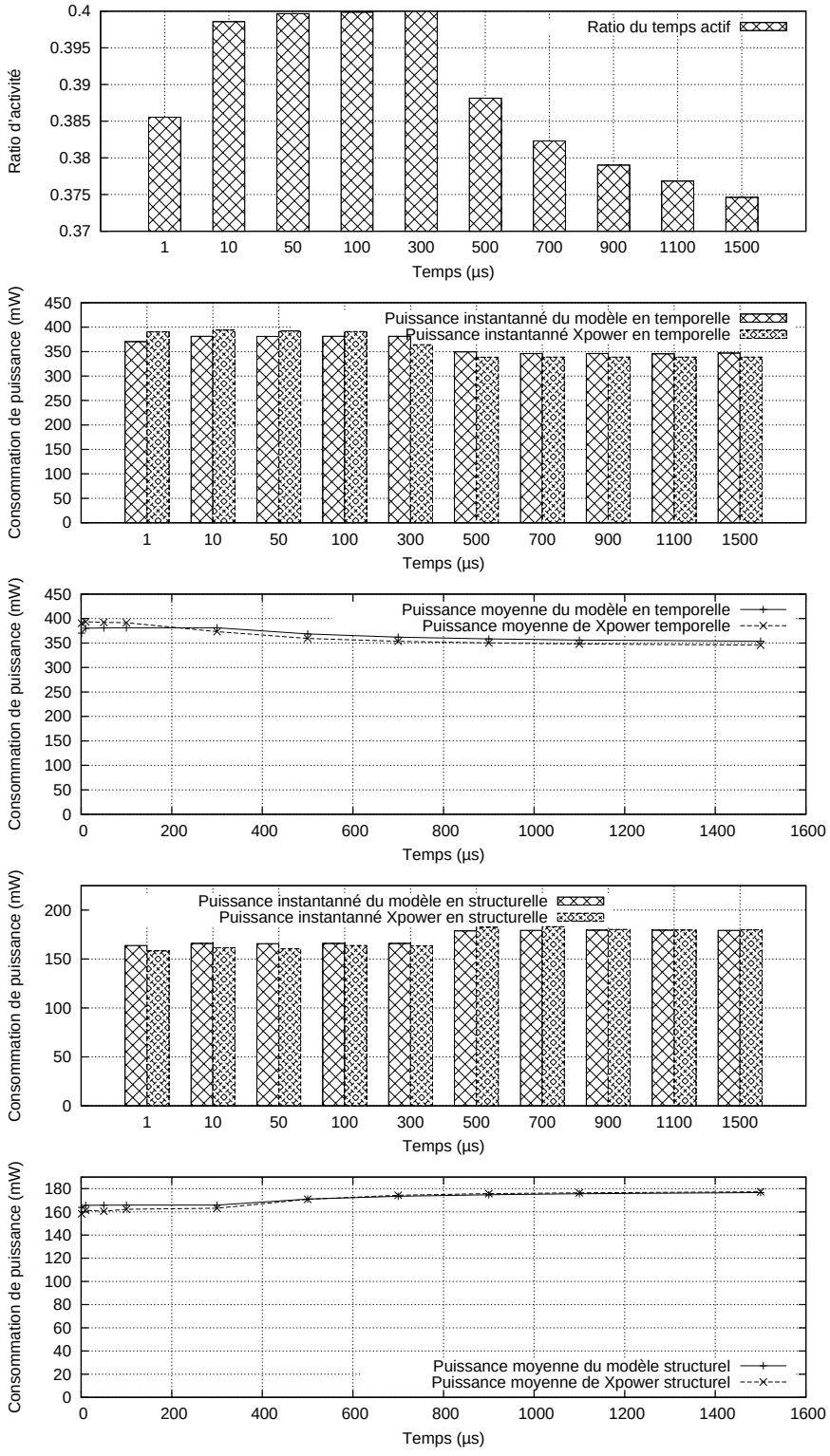


Figure 5.16 Consommation de puissance du Microblaze avec le Dhrystone à 12 ns

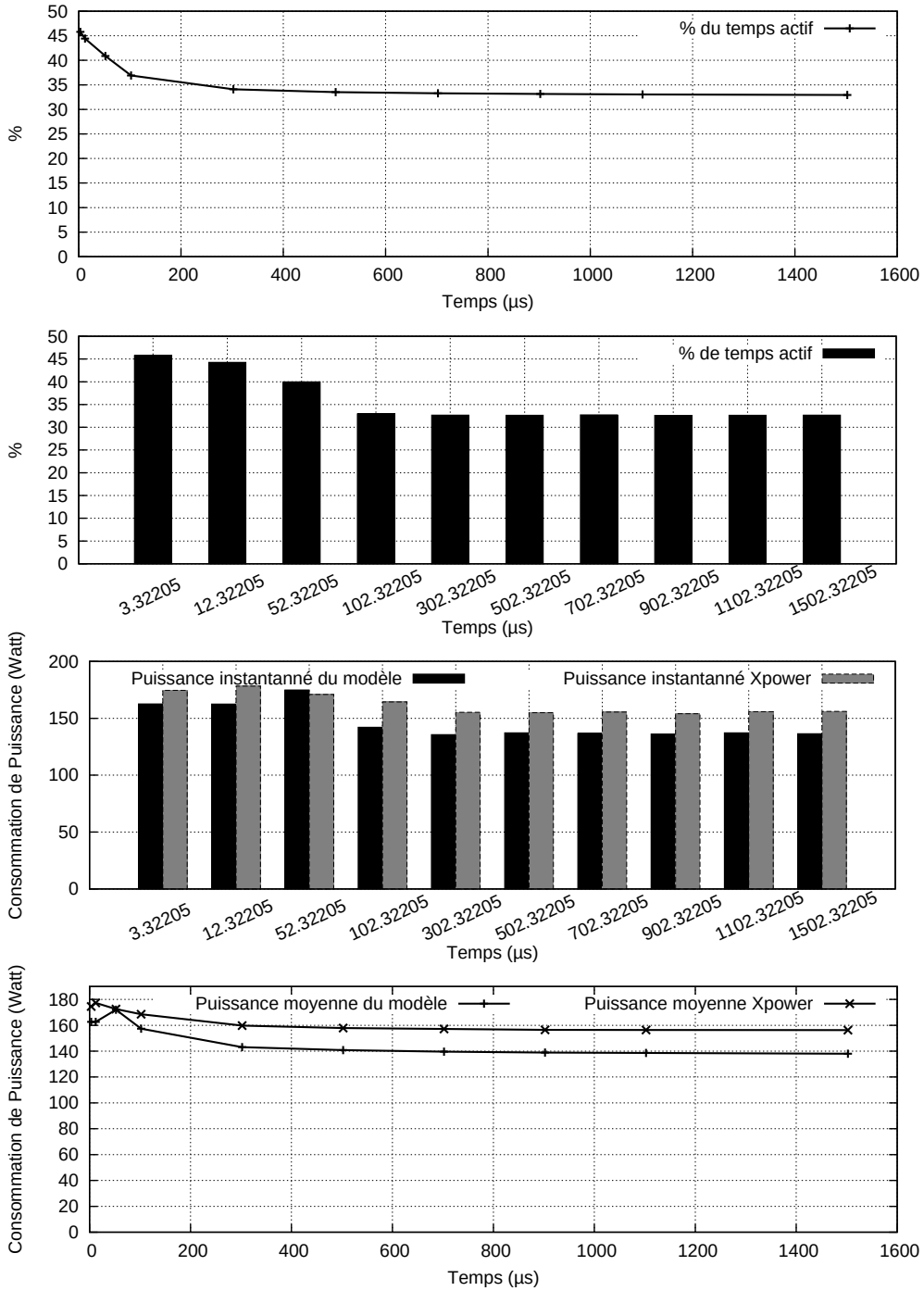


Figure 5.17 Consommation de puissance du Microblaze avec Coremark à 12ns

5.5 Bus

Lors des mesures du bus OPB, une attention particulière a été apportée aux activités parasites. La raison est que dans le cas du bus LMB le processeur envoyait des données lors d'une lecture et laissait passer l'adresse et la donnée au bus LMB qui est utilisé par un autre bus. De plus, le contrôleur de mémoire permettait à la BRAM qui était en écriture de renvoyer la donnée inscrite sur le bus.

Le cas du bus OPB est différent. La raison est que le bus est composé d'une logique en OR. Donc, une activité parasitique provenant du processeur causerait des erreurs de communication. Ceci demande à tous les IP qui ne doivent pas communiquer sur le bus d'être silencieux. De plus, le bus OPB ne permet pas d'envoyer de valeurs inutiles (comme envoyer une valeur durant une lecture ou recevoir une valeur durant une écriture). La raison est que les contrôleurs doivent être à zéro sinon les valeurs interféreront avec les valeurs transigées entre le maître et l'esclave (voir la Figure 5.18). Les valeurs de sortie de l'esclave sont aussi les valeurs d'entrée pour l'esclave. Ceci oblige les IP à n'envoyer que les valeurs utiles pour la communication sur le bus OPB. Ce comportement est celui qui est retrouvé habituellement dans les standards de bus.

Les contraintes que le bus OPB impose sur les communications permettent de limiter les valeurs inconnues comme dans le cas de la mémoire. Voici une mesure effectuée avec le Dhrystone avec 6 maîtres et 6 esclaves avec une période de 15 ns (voir la Figure 5.19 et Figure 5.20). Une autre simulation à 12 ns a été effectué avec 4 maîtres et 2 esclaves. Pour le Coremarks on a simulé avec une fréquence de 12 ns avec le processeur comme seul maître (sans bridge) et 4 esclaves.

La relation entre le taux d'activité et la consommation de puissance montre que la consommation durant un cycle mort est presque nulle. Dans la Figure 5.22 et Figure 5.21, il est possible de voir que la consommation est très faible quand il n'y a pas de communication. Dans ce cas, le Dhrystone a été exécuté une fois pour montrer la différence de consommation quand le processeur utilise une BRAM sur le bus OPB mesuré comme mémoire de donnée et quand le bus tombe silencieux.

La précision du modèle est excellente, l'erreur maximale en temporel est de 10% avec 6 maîtres et 6 esclaves avec une période d'horloge 15 ns. Avec l'architecture contenant 4 maîtres et 2 esclaves avec une période d'horloge 12 ns, l'erreur tombe à 0.8%. La précision du modèle est due au fait que le bus OPB n'a pas d'activité parasite et le bus est un IP qui est majoritairement une interconnexion avec une logique en OR. Le bus est un IP très régulier et le nombre de maîtres a peu d'impact sur l'architecture pertinente aux esclaves (et inversement). Donc, le modèle peut modéliser l'impact des maîtres et des esclaves séparément.

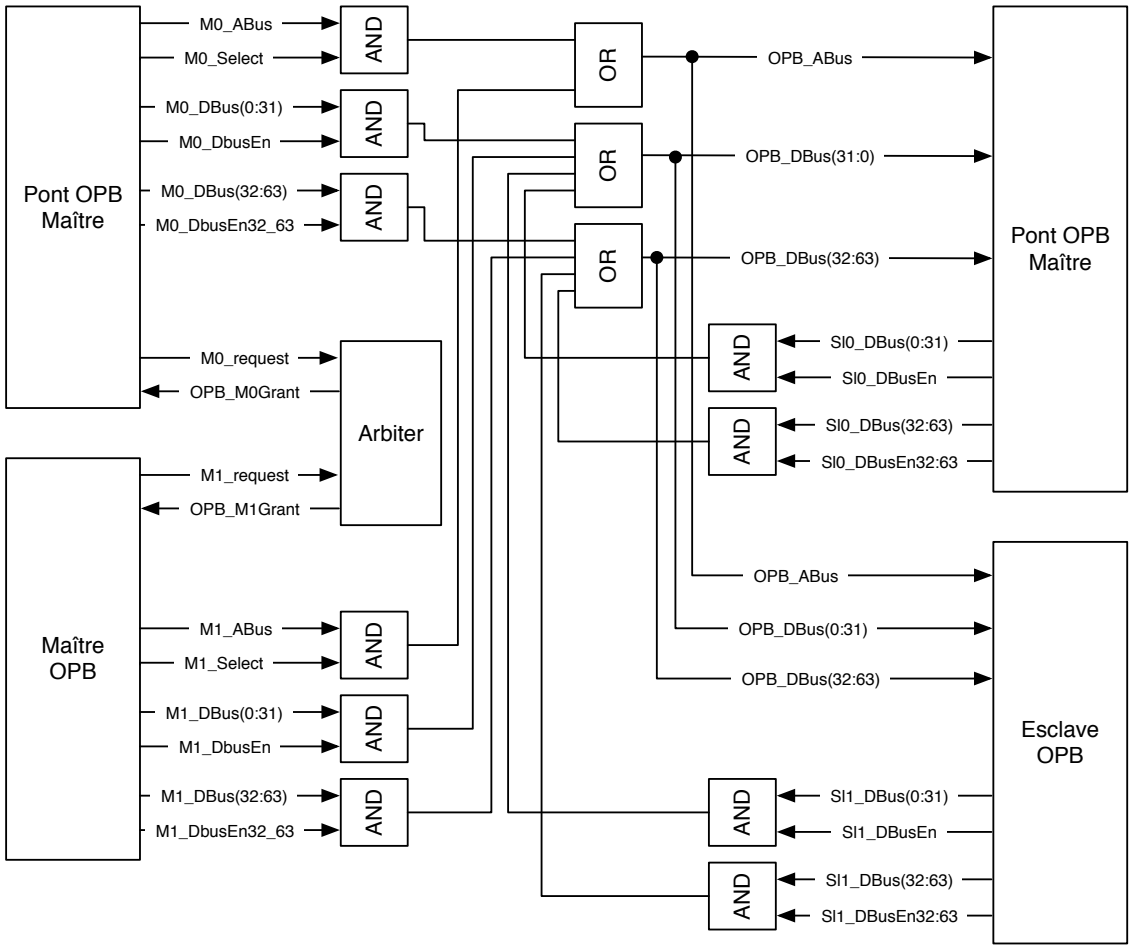


Figure 5.18 Implémentation du bus OPB[20]

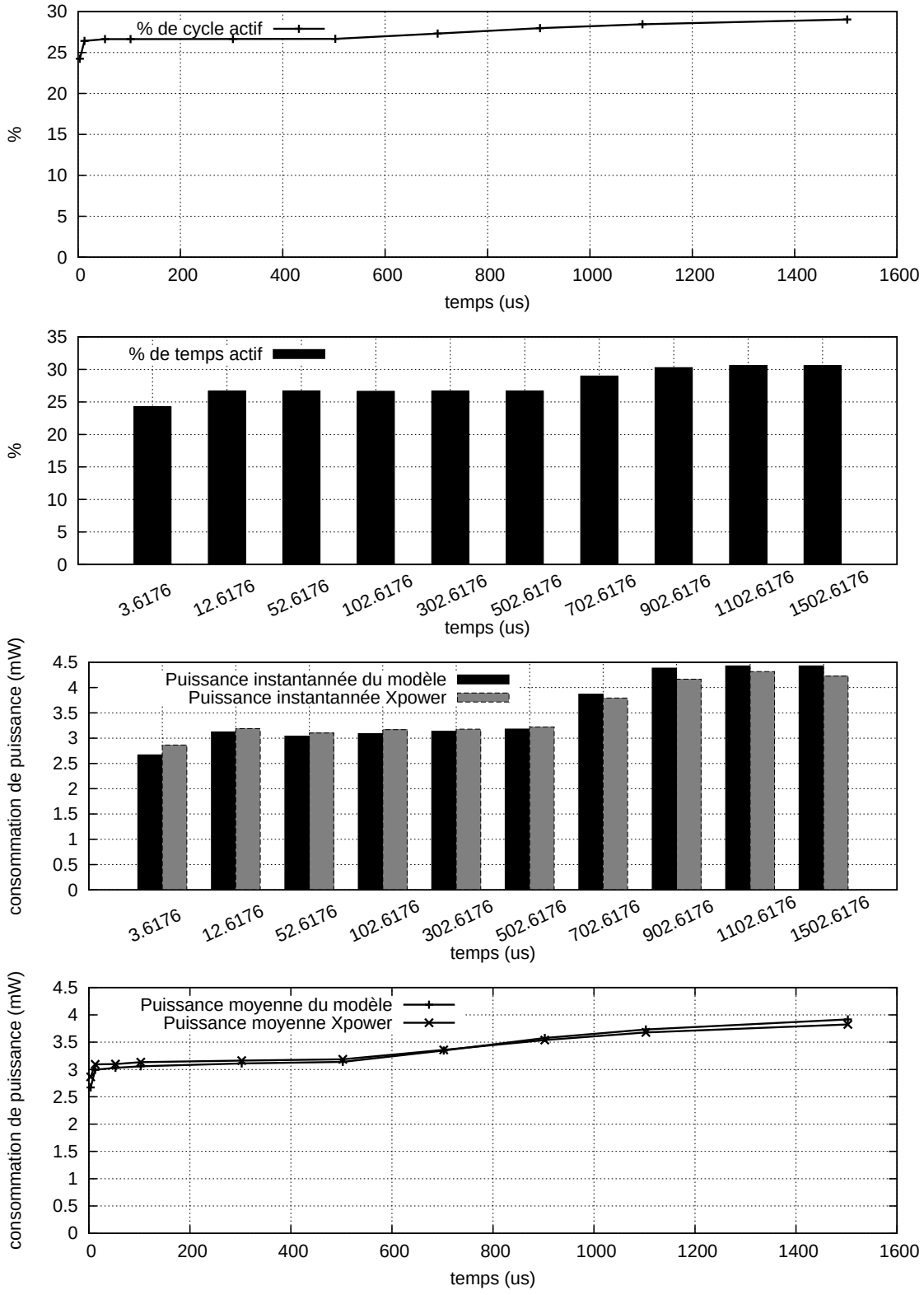


Figure 5.19 Consommation de six Maîtres et Esclaves en simulation structurelle avec le Dhrystone à 15 ns

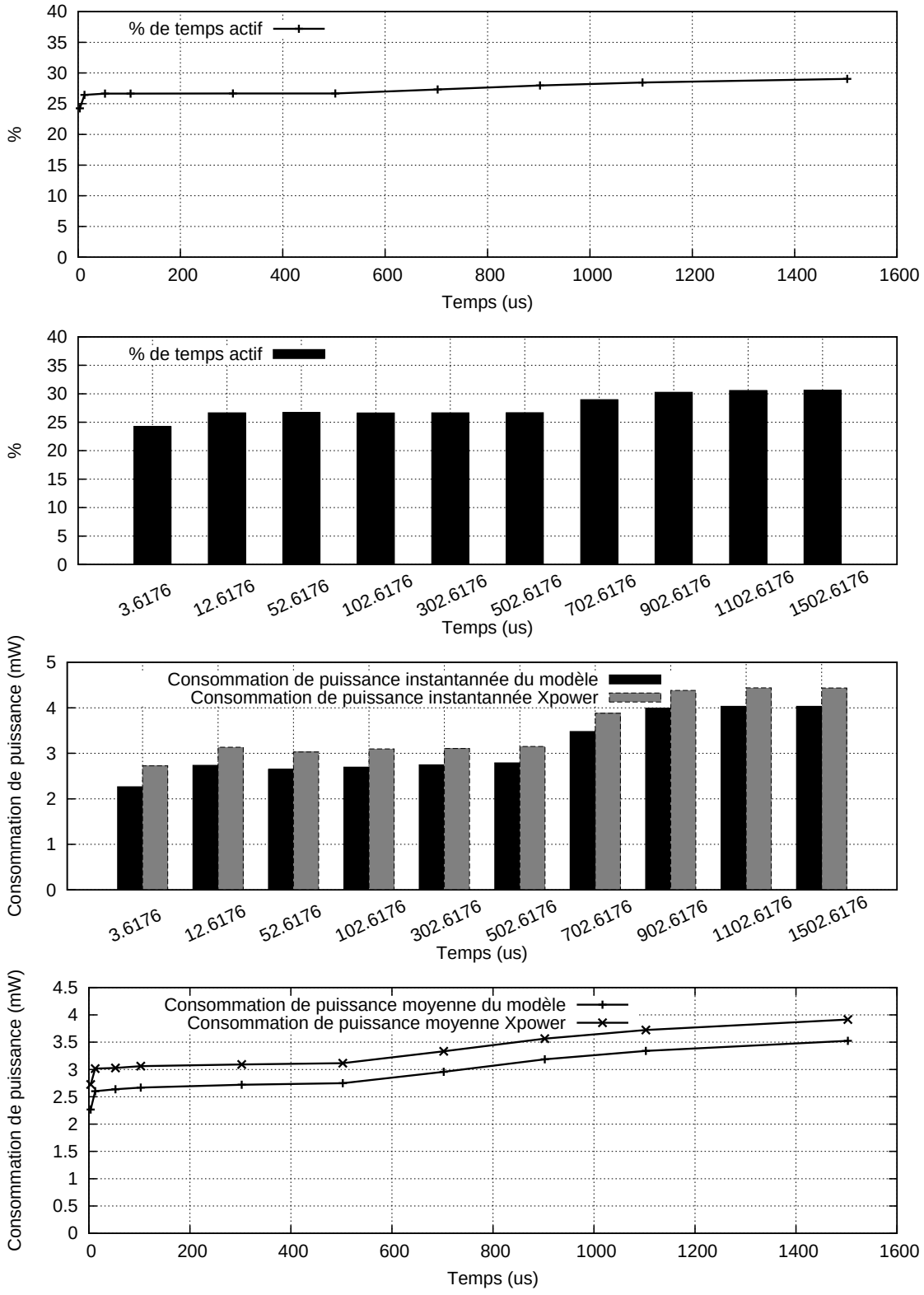


Figure 5.20 Consommation de six Maître et Esclave en simulation temporelle à une période de 15 ns avec le Dhrystone

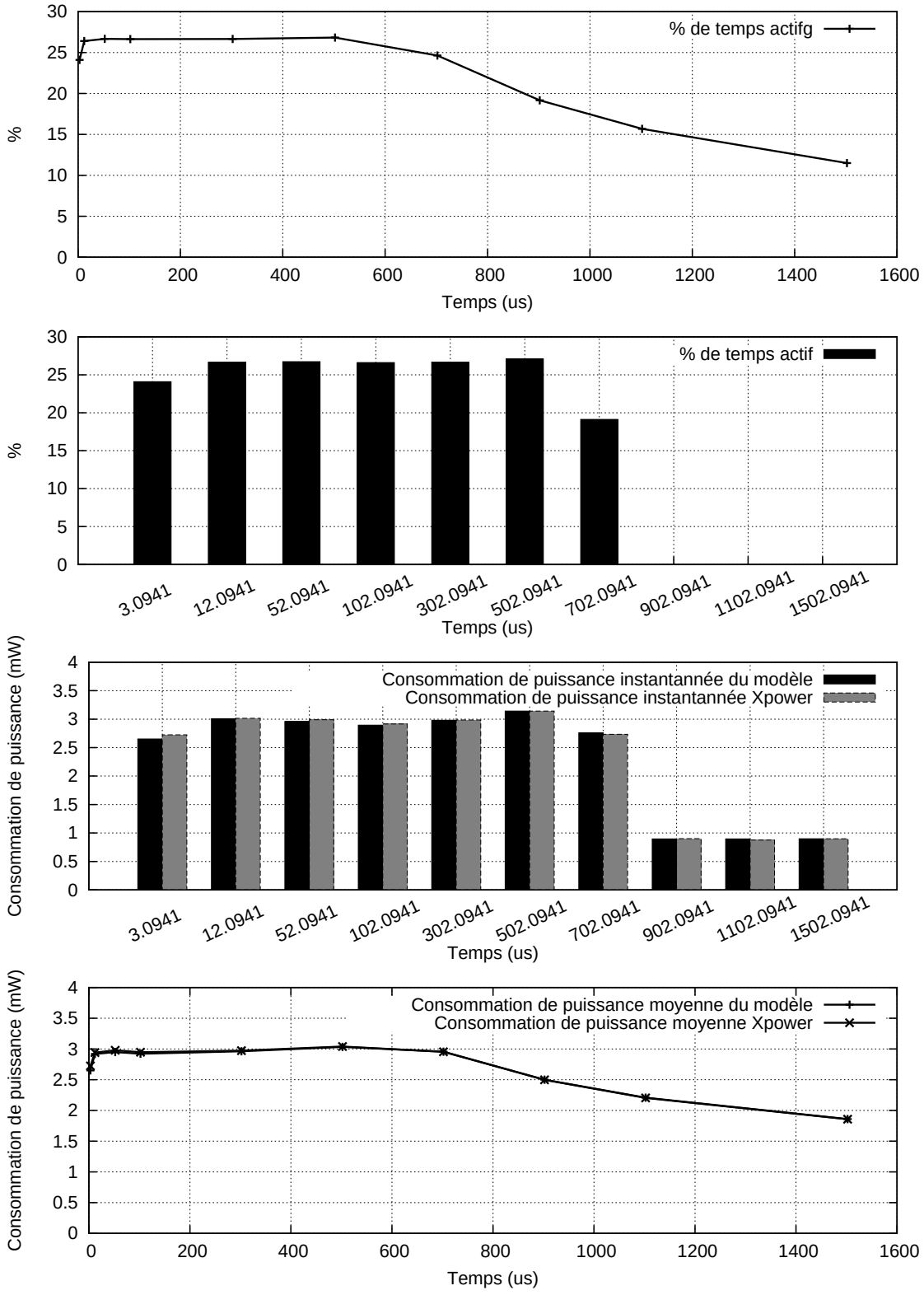


Figure 5.21 Consommation de quatre Maîtres et deux Esclaves en simulation structurelle pour le Dhrystone à 12 ns

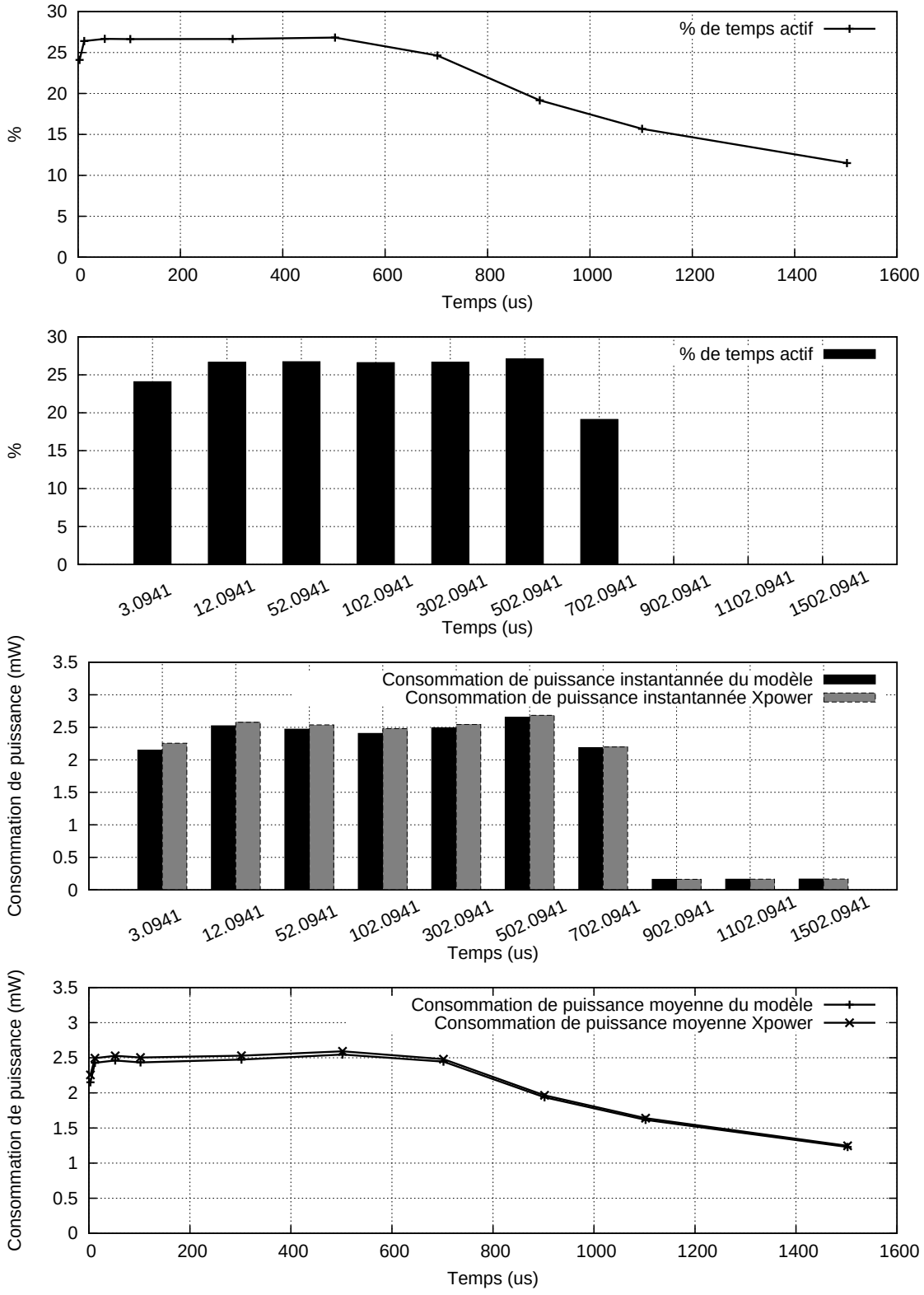


Figure 5.22 Consommation de quatre Maîtres et deux Esclaves en simulation temporelle pour le Dhrystone à 12 ns

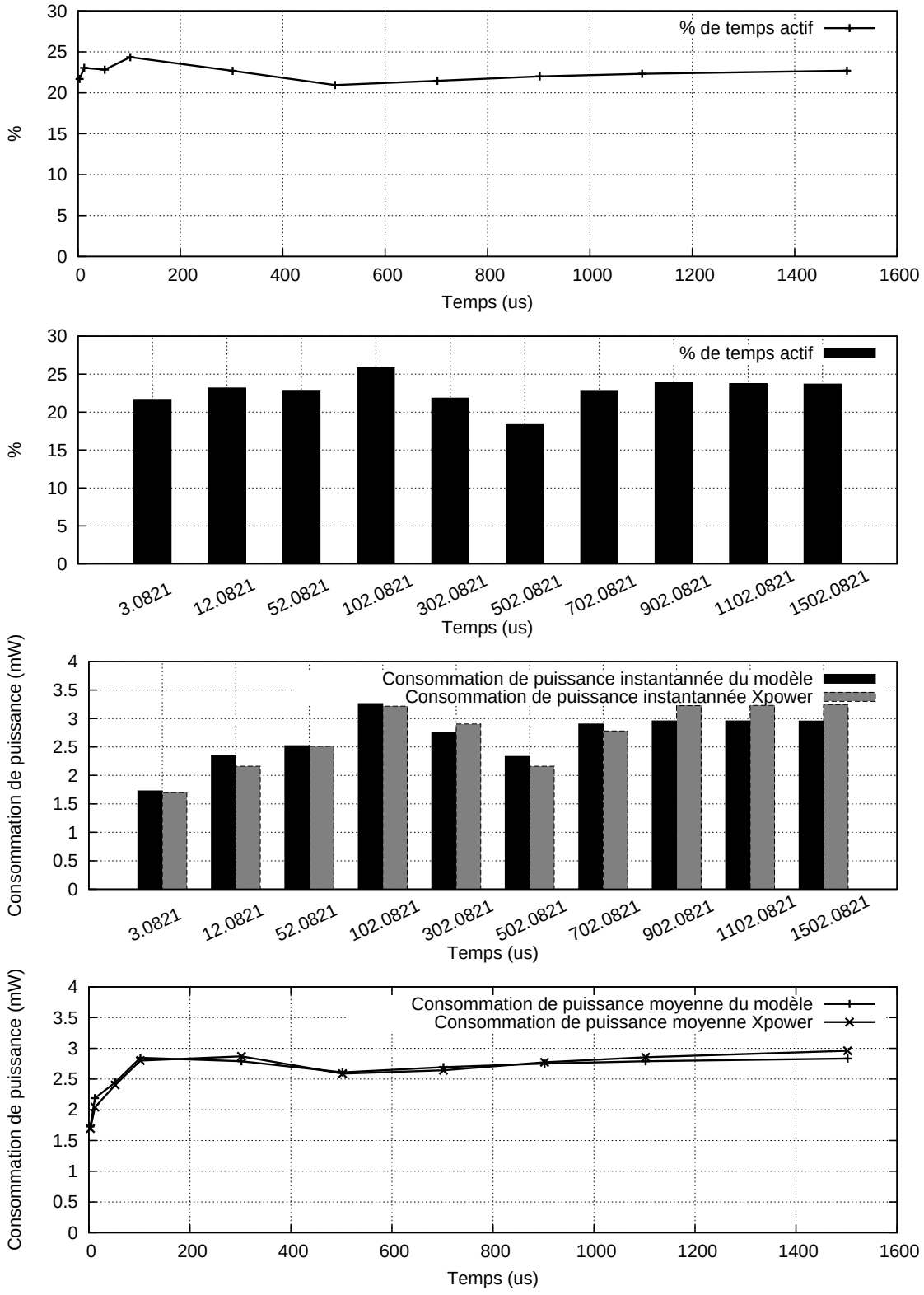


Figure 5.23 Consommation de un Maîtres et quatre Esclaves en simulation temporelle pour le Coremark à 12 ns

5.6 Résultat Global

Bien que l'estimation de la consommation d'un IP individuel puisse donner un bon résultat, il est important que l'ensemble des estimations donne un bon estimé. L'estimation globale de la consommation du système doit être proche de la consommation de puissance mesurée par l'outil Xpower. Les différentes mesures effectuées sont reportées dans le Tableau 5.1 pour présenter la précision globale de la méthode.

Tableau 5.1 Résumé des mesures moyennes des différents IP

	Dhrystone			Coremark		
	Xpower mW	Estimation mW	Delta %	Xpower mW	Estimation mW	Delta %
Compteur_0	5,6734004	5,1488389	-9,25	8,3506565	5,2195967	-37,5
Compteur_timer	5,5375061	4,8847929	-11,8	8,2535074	4,8847929	-40,8
PIC	2,3704801	0,3548189	-85,0	2,9534033	0,4320936	-85,4
Mémoire	188,4535880	157,6206760	-16,4	20,3032860	124,6769429	516,28
Processeur	345,6932837	353,6960085	2,31	156,3003240	138,0369914	-11,68
Bus	3,9129713	3,5233373	-9,95	2,9594052	2,8335248	-4,25
Globale	551,6412296	525,2284725	-4,79	199,1205824	276,0839423	38,65

Malgré certaines ratées de la méthode d'estimation, particulièrement dans le cas du PIC et de la mémoire avec le Coremark, les résultats globaux sont relativement précis. En considérant la consommation dans sa globalité, la précision est très bonne. La raison est que les IP qui ont la consommation la plus grande ont aussi la meilleure précision. Par contre, on ne peut négliger le fait que certains IP ont des erreurs plus grandes qui sont inacceptables. Il est intéressant de remarquer que les IP les plus complexes ont de meilleurs résultats, car leur PAR est le plus stable. La mémoire n'est pas du groupe, à cause des limitations de l'information disponible à cause de son comportement durant les cycles morts qui contrevient à la définition standard TLM (voir Section 4.5.2 et 5.3). Ceci est dû au fait que les IP les plus complexes ont aussi le plus d'interconnexions, ce qui limite les variations extrêmes dans leur PAR. Un IP volumineux en ressource est routé en premier pour obtenir de meilleures performances, par la suite les IP comme le PIC sont placés dans le FPGA dans les espaces restant, car ses contraintes sont plus faciles à rencontrer. Ceci fait que les IP qui sont moins priorisés dans le PAR auront plus de variabilité dans leur consommation. De plus, les IP ayant une estimation de puissance avec plus d'erreurs ont quand même une corrélation avec la mesure de Xpower très forte. Ce qui indique que l'équation reflète bien l'importance des différentes activités et charge de l'IP. Par contre, le PAR peut changer l'amplitude de la consommation par rapport aux mesures. Le facteur TP doit mieux incorporer le changement de PAR. Pour cela, il faudra considérer

l'apport de l'architecture et du taux d'utilisation des ressources du FPGA pour évaluer leur effet sur le placement et l'allongement du routage. Puisque notre analyse de l'implémentation se concentre majoritairement sur l'impact des paramètres de l'IP sur le PAR, le modèle ne tient pas en compte la globalité du système qui aura un effet sur les différentes composantes.

CHAPITRE 6

CONCLUSION

Les résultats obtenus montrent qu'en général les estimations sont bonnes. Malgré certaines erreurs dans les résultats, ils ne remettent pas en question la méthode, mais plutôt les choix effectués ou les particularités des IP fautifs.

Certains modèles ont eu des problèmes dans la précision de leurs résultats (particulièrement pour le PIC et la mémoire), par contre les estimations ont une bonne corrélation. Bien que certains résultats n'aient pas une précision acceptable, le comportement du modèle indique que l'impact du PAR est la cause de cette disparité. Seule l'amplitude de la consommation cause l'erreur. La relation de la consommation entre les différentes activités est similaire car les courbes ont une forte corrélation. De plus, l'erreur du modèle est beaucoup plus élevée dans le cas du banc de test Coremark. Pour la mémoire, les moins bons résultats viennent du fait que le processeur Microblaze cause de l'activité à la mémoire durant les cycles mort. Puisque les modèles utilisent une méthode TLM, pour fonctionner avec Space CoDesign, il est impossible au module de connaître cette activité. Dans le cas des minuteries, Coremark fait plus d'accès au compteur que le Dhrystone car il l'active plus tôt durant la simulation ; mais ceci n'explique pas la différence entre les estimations et les mesures de Xpower. Ceci tend à indiquer que le PAR a été fait différemment ce qui a causé une différence dans la consommation où le modèle estime incorrectement l'activité de la minuterie.

Le cas du PIC tend à corroborer cette hypothèse car celui-ci ne correspond pas aux deux bancs de test. Puisque les mesures montraient un comportement fortement linéaire et que les résultats divergent, le PAR affecte fortement les IP qui ont des contraintes de performance faible. Le PIC est un module qui ne fait que répéter les interruptions reçues, de ses multiples ports, au processeur, car ce dernier n'a qu'un port d'interruption. La contrainte de délai du synthétiseur sur le module est très facile à rencontrer même si les interconnexions sont très longues avec ce module. Alors la consommation augmentera drastiquement car les interconnexions du Virtex2 coûtent très cher en consommation [23, 41] causant 50% de la consommation. Pour pouvoir faire une estimation de consommation qui estime un PAR plus complexe il faut ajouter comme paramètre, l'architecture du système et l'utilisation des ressources du FPGA pour ne pas être limité à une vision de l'IP seul.

6.1 Synthèse des travaux

Ce mémoire a exploré une méthode qui subdivise l'analyse d'un IP en trois parties : l'activité ou action de l'IP qui définit un ensemble de transitions dans la logique et les connexions ; l'architecture du module qui définit ses caractéristiques et options de fonctionnement ; et finalement, les particularités d'implémentation de l'IP. Malgré des erreurs dans les estimations de certains modules, les résultats sont généralement positifs. De plus, l'estimation globale du système est plutôt bonne, bien qu'elle peut faire preuve de quelques améliorations.

La méthode présentée au cours de ce mémoire a été appliquée à cinq IP différents tant dans leur architecture que dans leur fonction. La méthode a été capable de produire des estimations qui ont de fortes corrélations malgré le fait que les résultats n'étaient pas toujours aussi précis qu'attendu. Ceci soutient que la méthode est valable, pour certains IP (particulièrement le PIC et la minuterie) il faut ajouter un paramètre permettant de tenir compte du PAR. Pour le processeur et le bus, les excellents résultats montrent que la méthode fonctionne bien quand les IP sont de plus grosse taille.

La mémoire utilisée localement avec le Microblaze se situe dans une situation où l'activité parasite qui ne pouvait pas être estimée par le modèle (car elle n'est pas disponible par une simulation TLM) cause une erreur assez importante. La méthode présentée ne pouvait combler le manque d'information qui dépendait d'une décision d'implémentation causant une activité non documentée dans le comportement du Microblaze.

La méthode permet d'accélérer la production d'un modèle d'IP pour effectuer des estimations de consommation de puissance durant une simulation haut-niveau TLM. De plus, en utilisant une simulation haut-niveau, la méthode permet d'obtenir des simulations très rapides et elle permet de ne pas ralentir la simulation à cause de la prise de donnée [40]. La simulation et le profilage en TLM avec l'outil Space Codesign a été fait en 8,9 secondes tandis que l'analyse de consommation de puissance avec Modelsim et Xpower a durée 73665 secondes. Ceci donne une accélération de trois ou quatre ordre de grandeur.

6.2 Améliorations futures

Les résultats indiquent que les modèles développés peuvent être améliorés pour obtenir des résultats encore plus précis et élargir les options d'application de cette méthode. Bien que cette méthode utilise plusieurs scripts pour automatiser la création des modèles, il y a encore quelques étapes manuelles. Il faut encore récupérer les résultats et les importer manuellement dans un outil pour déterminer les valeurs de l'équation. Le format de l'équation est encore déterminé manuellement et ajusté à la main lorsque les résultats ne sont pas satisfaisants. À ce jour, un projet de recherche qui a débuté pour répondre à ce problème. Il permet-

tra de définir une structure pour l'analyse automatisée d'un IP et plus particulièrement les processeurs. Il sera possible avec ce projet de définir le jeu d'instructions et les options du processeur et obtenir un modèle précis et portable. En permettant d'automatiser l'ensemble de l'analyse, la création du modèle et la vérification de la précision, il sera plus facile de faire des modèles des différents processeurs pour plusieurs technologies de FPGA vendus dans le commerce. Pour assister la modélisation automatique des processeurs, il serait pertinent de réutiliser les travaux de Brandolese, Fornaciari, Salice et Sciuto [8]. Leur méthode représente la consommation des différentes instructions par l'activation des différentes étapes d'un pipeline (décodage, exécution, etc...). Ceci leur permet d'estimer la consommation d'une instruction non mesurée en estimant sa consommation avec les étapes du pipeline utilisé. De plus, ils peuvent ajouter un nouveau processeur car, bien que la consommation de courant absolue des deux processeurs est différente pour une même instruction, la valeur relative est de la même amplitude. En effectuant un minimum de mesure, le modèle peut être adapté pour un nouveau processeur en obtenant une erreur de $-5\% \pm 10\%$. L'avantage est que la méthode s'adapte bien à celle utilisée dans ce travail.

Pour limiter l'erreur que le PAR de l'architecture cause dans les estimations, il est possible de se servir d'un modèle d'estimation de routage. Un de ces modèle permet d'estimer la capacité d'une connexion à la suite d'un placement sans avoir effectué le routage [2]. Les IP ont une structure bien définie, et l'implémentation de cet IP ne changera pas trop si les contraintes de délais sur les signaux sont sévères et que peu de ressources du FPGA sont utilisés. Ce modèle peut être réutilisé à plus haut-niveau pour obtenir un facteur d'augmentation de charge capacitive de routage pour un système qui utilise une quantité plus grande de ressources matérielles où l'architecture du système force certains IP à augmenter leur routage (comme dans le cas du PIC). Au lieu de se fier au placement de la synthèse, on pourra utiliser l'estimation d'utilisation de ressource (cellule logique, mémoire, etc...). Cette information est fournie dans la documentation des IP. L'architecture du système pourra être tenue en compte dans l'outil pour estimer un facteur de charge de chacun des IP basés sur la probabilité que leur routage et leur consommation augmentent. Ainsi, le changement d'architecture affectera la consommation des différents IP et permettra de diminuer l'erreur qui a été observée pour les modules comme la minuterie et le PIC.

Dans la même veine d'idée, il est possible d'utiliser un modèle pour estimer l'impact de l'architecture du FPGA sur la consommation de l'IP. Un de ces modèle permet de prendre les caractéristiques du FPGA et d'estimer la consommation de puissance dynamique, de fuite et de court-circuit [39]. La possibilité de modéliser la consommation de puissance de plusieurs FPGA sur un même modèle permet l'utilisation du modèle pour effectuer l'ajout d'un nouveau FPGA dans la librairie sans devoir refaire toutes les mesures. Il suffit de prendre

un IP portable sur plusieurs plateformes et d'utiliser ce modèle pour adapter les mesures d'un ou plusieurs FPGA à une nouvelle technologie. Il suffit par la suite de faire quelques mesures de vérifications pour s'assurer de la précision des estimations.

Avec ces ajouts, le modèle peut devenir un outil très performant pour le profilage et partitionnement automatique d'un système [31]. Le partitionnement et le profilage de l'article étaient effectués principalement sur la performance et la quantité de ressources utilisées. Avec un estimateur de puissance et les améliorations spécifiées précédemment, il serait possible d'explorer un grand nombre d'IP sur plusieurs FPGA en même temps pour sélectionner la meilleure solution en terme de performance et ressource, mais aussi en terme de consommation. Cet outil permettra l'exploration d'architectures, sans être limité par la technologie et le temps d'ajout d'implémentation des IP disponible, et permettra à l'utilisateur de faire un choix plus judicieux pour son application.

RÉFÉRENCES

[1] AHMADINIA, A., AHMAD, B. et ARSLAN, T. (2008). Efficient high-level power estimation for multi-standard wireless systems. *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*. 275 –280.

[2] ANDERSON, J. H. et NAJM, F. N. (2004). Interconnect capacitance estimation for FPGAs. *ASP-DAC '04 : Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. IEEE Press, Piscataway, NJ, USA, 713–718.

[3] ANTON, M., COLONESCU, I., MACII, E. et PONCINO, M. (2001). Fast characterization of rtl power macromodels. *Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on*. vol. 3, 1591 –1594 vol.3.

[4] ARSHAK, K., JAFER, E. et IBALA, C. (2007). Power testing of an FPGA based system using modelsim code coverage capability. *Design and Diagnostics of Electronic Circuits and Systems*, 0, 1–4.

[5] BAILEY, B. et MARTIN, G. (2010). Codesign experiences based on a virtual platform.

[6] BANSAL, N., LAHIRI, K. et RAGHUNATHAN, A. (2007). Automatic power modeling of infrastructure IP for system-on-chip power analysis. *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*. 513 –520.

[7] BEUCHER, N., BELANGER, N., SAVARIA, Y. et BOIS, G. (2007). A methodology to evaluate the energy efficiency of application specific processors. *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*. 983 –986.

[8] BRANDOLESE, C., FOMACIAN, W., SALICE, F. et SCIUTO, D. (2000). An instruction-level functionality-based energy estimation model for 32-bits microprocessors. *Design Automation Conference, 2000. Proceedings 2000. 37th*. 346 –350.

[9] CAI, L. et GAJSKI, D. (2003). Transaction level modeling : an overview. *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*. 19 – 24.

[10] CHO, Y., KIM, Y., PARK, S. et CHANG, N. (2008). System-level power estimation using an on-chip bus performance monitoring unit. *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*. 149 –154.

[11] CLARKE, J., GAFFAR, A., CONSTANTINIDES, G. et CHEUNG, P. (2006). Fast word-level power models for synthesis of FPGA-based arithmetic. *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. 4 pp.

[12] COBURN, J., RAVI, S. et RAGHUNATHAN, A. (2005). Power emulation : a new paradigm for power estimation. *Design Automation Conference, 2005. Proceedings. 42nd*. 700 – 705.

[13] COREMARK (2010). Coremark.
http ://www.coremark.org/home.php.

[14] ELLEOUET, D., JULIEN, N., HOUZET, D., COUSIN, J.-G. et MARTIN, E. (2004). Power consumption characterization and modeling of embedded memories in Xilinx Virtex 400e FPGA. *DSD '04 : Proceedings of the Digital System Design, EUROMICRO Systems*. IEEE Computer Society, Washington, DC, USA, 394–401.

[15] FORTE DESIGN SYSTEMS (2011). Cynthesizer.
http ://www.forteds.com/products/cynthesizer.asp.

[16] GERSTLAUER, A., HAUBELT, C., PIMENTEL, A., STEFANOV, T., GAJSKI, D. et TEICH, J. (2009). Electronic system-level synthesis methodologies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28, 1517 –1530.

[17] GHODRAT, M., LAHIRI, K. et RAGHUNATHAN, A. (2007). Accelerating system-on-chip power analysis using hybrid power estimation. *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*. 883 –886.

[18] GRAPHICS, M. (2007). *ModelSim® SE Reference Manual*. Mentor Graphics, 8005 S.W. Boeckman Road, Wilsonville, Oregon, 6th édition.
http ://model.com/content/modelsim-se-high-performance-simulation-and-debug.

[19] GRAPHICS, M. (2007). *ModelSim® SE User's Manual*. Mentor Graphics, 8005 S.W. Boeckman Road, Wilsonville, Oregon, 6th édition.
http ://model.com/content/modelsim-se-high-performance-simulation-and-debug.

[20] IBM (2001). On-chip peripheral bus. Specification SA-14-2528-02, IBM.

[21] JEVTIC, R., CARRERAS, C. et CAFFARENA, G. (2008). Fast and accurate power estimation of FPGA DSP components based on high-level switching activity models. *Int. J. Electron. (UK)*, 95, 653 – 668.

[22] JHA, N., RAGHUNATHAN, A., LAKISHMINARAYANA, G. et TAN, T. (2001). High-level software energy macro-modeling. *DAC '01 : Proceedings of the 38th annual Design Automation Conference*. ACM, 605–610.

[23] JULIEN, N. (2006). Architecture reconfigurable faible consommation. *Conception faible consommation de système temps réel*. ECoFac.
http ://www.i3s.unice.fr/ECoFaC/PDF/julien/FPGAconsoEcofac.pdf.

[24] KLEIN, F., LEAO, R., ARAUJO, G., SANTOS, L. et AZEVEDO, R. (2009). A multi-model engine for high-level power estimation accuracy optimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17, 660 –673.

[25] LEE, H. G., LEE, K., CHOI, Y. et CHANG, N. (2005). Cycle-accurate energy measurement and characterization of FPGAs. *Analog Integrated Circuits and Signal Processing*, 42, 239–251.

[26] LEE, J., VINNAKOTA, B. et LUCKE, L. (1998). Power estimation using input/output transition anaylsis (iota). *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*. vol. 6, 49 –52.

[27] LIU, D. et SVENSSON, C. (1994). Power consumption estimation in CMOS VLSI chips. *Solid-State Circuits, IEEE Journal of*, 29, 663 –670.

[28] MENTOR GRAPHICS (2011). Catapult C Synthesis.
http ://www.mentor.com/esl/catapult/overview.

[29] MOCAT (1998). L'algorithme de viterbi.
http ://www.irisa.fr/cosi/MOCAT/Travaux/Viterbi/top.html.

[30] MOORE, G. E. (2006). Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Newsletter, IEEE*, 20, 33 –35.

[31] MOSS, L. (2010). *Profilage, caractérisation et partitionnement fonctionnel dans une plate-forme de conception de systèmes embarqués*. Mémoire de maîtrise, Écooe Polytechnique.

[32] MOSS, L., CANTIN, M.-A., BOIS, G. et ABOULHAMID, E. M. (2008). Automation of communication refinement and hardware synthesis within a system-level design methodology. *RSP '08 : Proceedings of the 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*. IEEE Computer Society, Washington, DC, USA, 75–81.

[33] MOSS, L., DE NANCLAS, M., FILION, L., FONTAINE, S., BOIS, G. et ABOULHAMID, M. (2007). Seamless hardware/software performance co-monitoring in a codesign simulation environment with RTOS support. *DATE '07 : Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, San Jose, CA, USA, 876–881.

[34] NAJM, F. (1993). Transition density : a new measure of activity in digital circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 12, 310 –323.

[35] NAJM, F. (1997). Low-power design methodology : power estimation and optimization. *Circuits and Systems, 1997. Proceedings of the 40th Midwest Symposium on*. vol. 2, 1124 – 1129 vol.2.

[36] NAJM, F. N. (1995). Power estimation techniques for integrated circuits. *ICCAD '95 : Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society, Washington, DC, USA, 492–499.

[37] OU, J. et PRASANNA, V. K. (2006). Rapid energy estimation for hardware-software codesign using FPGAs. *EURASIP J. Embedded Syst.*, 2006, 17–28.

[38] PILATO, C., FERRANDI, F. et SCIUTO, D. (2011). A design methodology to implement memory accesses in high-level synthesis. *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*. 49 –58.

[39] POON, K. K., YAN, A. et WILTON, S. J. (2002). *Field-Programmable Logic and Applications : Reconfigurable Computing Is Going Mainstream*, Springer Berlin / Heidelberg, vol. 2438 de *Lecture Notes in Computer Science*, chapitre A Flexible Power Model for FPGAs. 312–321.

[40] ROGERS-VALLEE, M., CANTIN, M.-A., MOSS, L. et BOIS, G. (2010). IP characterization methodology for fast and accurate power consumption estimation at transactional level model. *Computer Design (ICCD), 2010 IEEE International Conference on*. 534 –541.

[41] SHANG, L., KAVIANI, A. S. et BATHALA, K. (2002). Dynamic power consumption in Virtex-ii FPGA family. *FPGA '02 : Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. ACM, New York, NY, USA, 157–164.

[42] SPEERS, T. (2010). Public enemies.
http ://chipdesignmag.com/lpd/speers/2010/09/09/public-enemies/.

[43] TIWARI, V., MALIK, S., WOLFE, A. et LEE, M. T.-C. (1996). Instruction level power analysis and optimization of software. *The Journal of VLSI Signal Processing*, 13, 223–238.

[44] WALL, L. (2002). The perl programming language.
http ://www.perl.org/.

[45] WALPOLE, R. E., MYERS, R. H., MYERS, S. L. ET YE, K. E. (2010). *Probability and Statistics for Engineers and Scientists*. Pearson, 9th édition.

[46] WANG, L., FRENCH, M., DAVOODI, A. et AGARWAL, D. (2006). FPGA dynamic power minimization through placement and routing constraints. *EURASIP J. Embedded Syst.*, 2006, 7–7.

[47] WANG, L., OLBRICH, M., BARKE, E., BUCHNER, T., BUHLER, M. et PANITZ, P. (2011). A theoretical probabilistic simulation framework for dynamic power estimation. *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*. 708 –715.

[48] WEICKER, R. P. (1984). Dhrystone : A synthetic systems programming benchmark. *Communications of the ACM*, 27, 1013 – 1030.

[49] WIMAX.COM BROADBAND SOLUTIONS INC (2011). Wimax.
http ://www.wimax.com/.

[50] WOLLER, J. (1996). The basics of monte carlo simulations.
http ://www.chem.unl.edu/zeng/joy/mclab/mcintro.html.

[51] XILINX (2006).
OPB Timer/Counter (v1.00b). Product Specification DS465, Xilinx.

[52] XILINX (2007). *Virtex-II Platform FPGA User Guide*. Xilinx, v2.2 édition.
http ://www.xilinx.com/support/documentation/user_guides/ug002.pdf.

[53] XILINX (2007). *Virtex-II Platform FPGAs : Complete Data Sheet*. Xilinx, v3.5 édition.
http ://www.xilinx.com/support/documentation/data_sheets/ds031.pdf.

[54] XILINX (2008). *Embedded System Tools Reference Manual*. Xilinx, edk 10.1, service pack 3 édition.
http ://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf.

[55] XILINX (2008). *MicroBlaze Processor Reference Guide*. Xilinx, v9.0 édition.
http ://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf.

[56] XILINX (2008). *OS and Lbraries Document Collection*. Xilinx, edk 10.1, service pack 3 édition.
http ://www.xilinx.com/support/documentation/sw_manuals/edk10_oslib_rm.pdf.

[57] XILINX (2010). Logicore IP on-chip peripheral bus v2.0 with opb arbiter. Product Specification DS402, Xilinx.

[58] XILINX (2010). Xpower analyzer overview.
http ://www.xilinx.com/itp/xilinx10/isehelp/isehelp_start.htm#xpa_c_overview.htm.

[59] XILINX (2012). Virtex-7 FPGA family.
http ://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm.